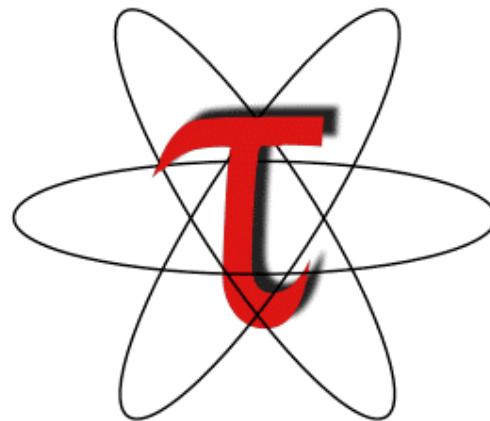


TAU Performance System

Sameer Shende, Allen D. Malony, Alan Morris
University of Oregon

{sameer, malony, amorris}@cs.uoregon.edu

ACTS Workshop, LBNL, Aug 25, 2006



UNIVERSITY
OF OREGON



Outline of Talk

- Overview of TAU
- Instrumentation
- Optimization of Instrumentation
- Measurement
- Analysis: ParaProf, Jumpshot and Vampir/VNG
- Future work and concluding remarks

TAU Performance System



- Tuning and Analysis Utilities (14+ year project effort)
- Performance system framework for HPC systems
 - Integrated, scalable, flexible, and parallel
- Targets a general complex system computation model
 - *Entities*: nodes / contexts / threads
 - Multi-level: system / software / parallelism
 - Measurement and analysis abstraction
- Integrated toolkit for performance problem solving
 - Instrumentation, measurement, analysis, and visualization
 - Portable performance profiling and tracing facility
 - Performance data management and data mining
- <http://www.cs.uoregon.edu/research/tau>



Definitions – Profiling

□ Profiling

- Recording of summary information during execution
 - inclusive, exclusive time, # calls, hardware statistics, ...
- Reflects performance behavior of program entities
 - functions, loops, basic blocks
 - user-defined “semantic” entities
- Very good for low-cost performance assessment
- Helps to expose performance bottlenecks and hotspots
- Implemented through
 - **sampling**: periodic OS interrupts or hardware counter traps
 - **instrumentation**: direct insertion of measurement code



Definitions – Tracing

Tracing

- Recording of information about significant points (**events**) during program execution
 - entering/exiting code region (function, loop, block, ...)
 - thread/process interactions (e.g., send/receive message)
- Save information in **event record**
 - timestamp
 - CPU identifier, thread identifier
 - Event type and event-specific information
- **Event trace** is a time-sequenced stream of event records
- Can be used to reconstruct dynamic program behavior
- Typically requires code instrumentation

Event Tracing: Instrumentation, Monitor, Trace

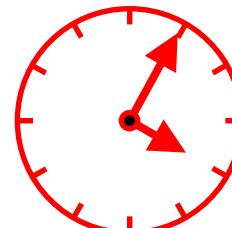


CPU A:

```
void master {  
    trace(ENTER, 1);  
    ...  
    trace(SEND, B);  
    send(B, tag, buf);  
    ...  
    trace(EXIT, 1);  
}
```

CPU B:

```
void slave {  
    trace(ENTER, 2);  
    ...  
    recv(A, tag, buf);  
    trace(RECV, A);  
    ...  
    trace(EXIT, 2);  
}
```



timestamp

MONITOR

Event definition

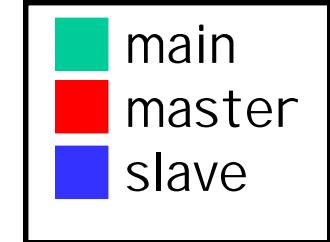
1	master
2	slave
3	...

...				
58	A	ENTER	1	
60	B	ENTER	2	
62	A	SEND	B	
64	A	EXIT	1	
68	B	RECV	A	
69	B	EXIT	2	
...				

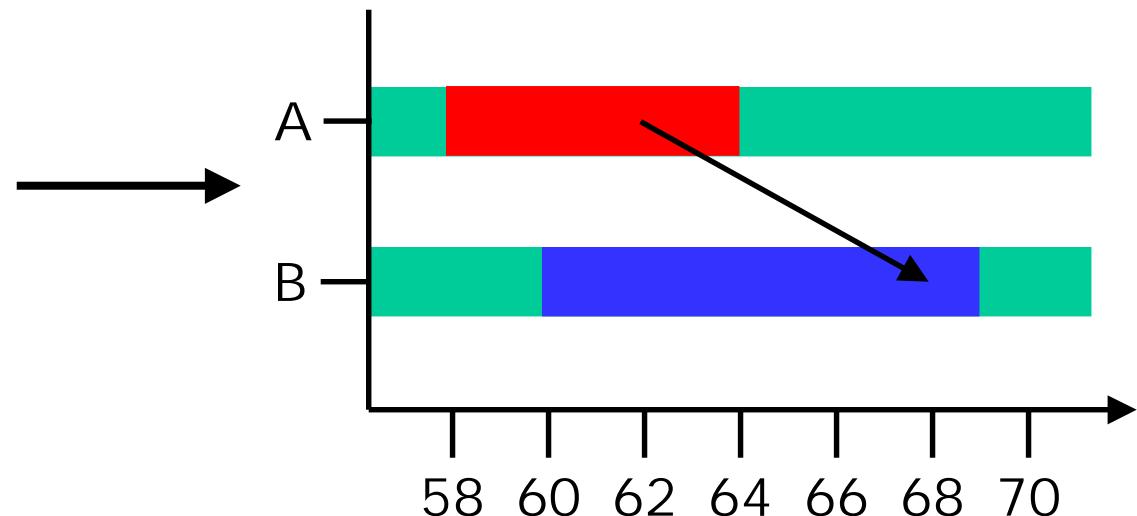
Event Tracing: “Timeline” Visualization



1	master
2	slave
3	...



...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			





TAU Parallel Performance System Goals

- Multi-level performance instrumentation
 - Multi-language automatic source instrumentation
- Flexible and configurable performance measurement
- Widely-ported parallel performance profiling system
 - Computer system architectures and operating systems
 - Different programming languages and compilers
- Support for multiple parallel programming paradigms
 - Multi-threading, message passing, mixed-mode, hybrid
- Support for performance mapping
- Support for object-oriented and generic programming
- Integration in complex software, systems, applications



Using TAU: A brief Introduction

- To instrument source code:

```
% setenv TAU_MAKEFILE  
$TAUROOTDIR/rs6000/lib/Makefile.tau-mpi-pdt
```

And use tau_f90.sh, tau_cxx.sh or tau_cc.sh as Fortran, C++ or C compilers:

```
% mpixlf90 foo.f90
```

changes to

```
% tau_f90.sh foo.f90
```

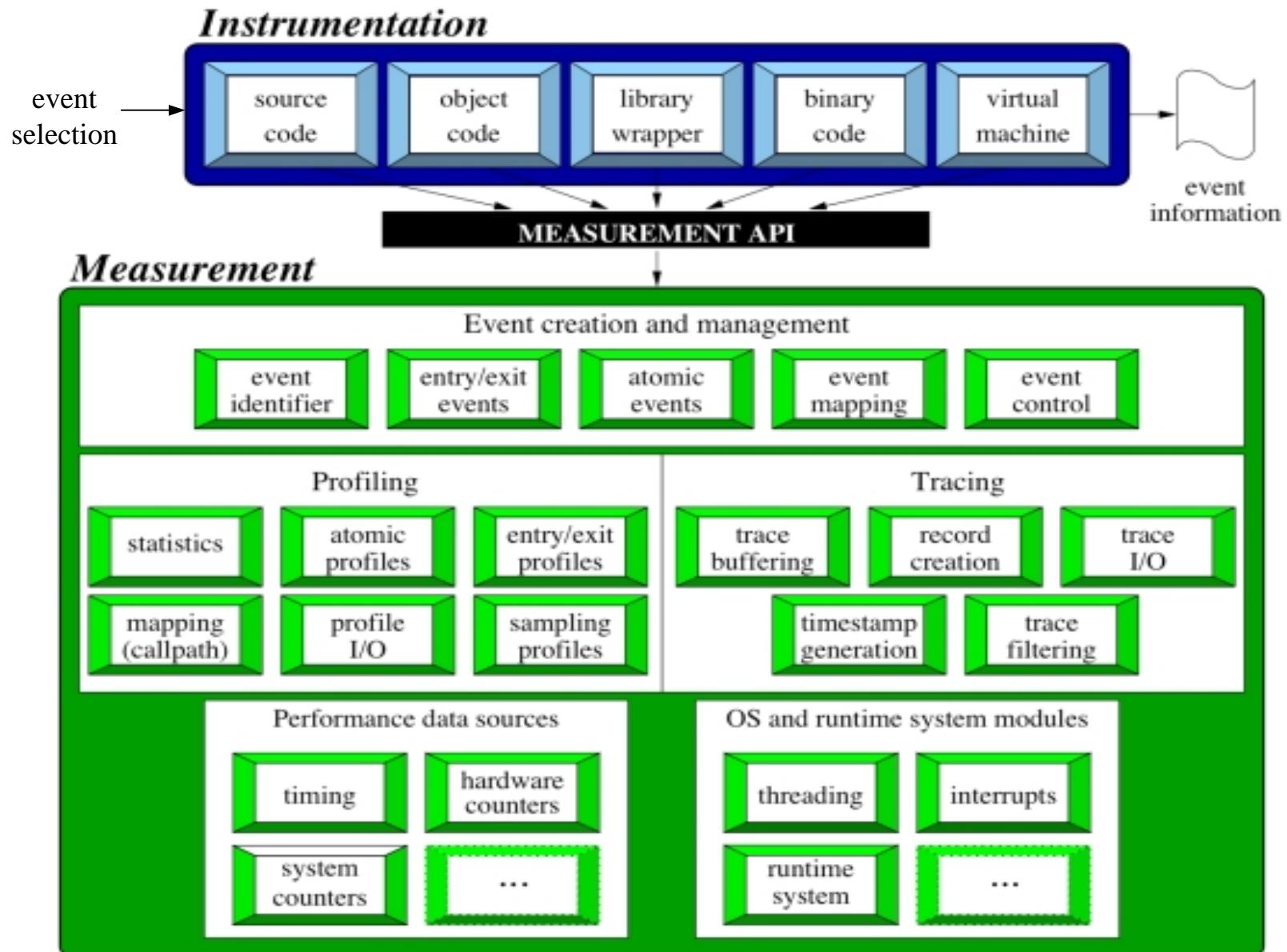
- Execute application and then run:

```
% pprof (for text based profile display)
```

```
% paraprof (for GUI)
```

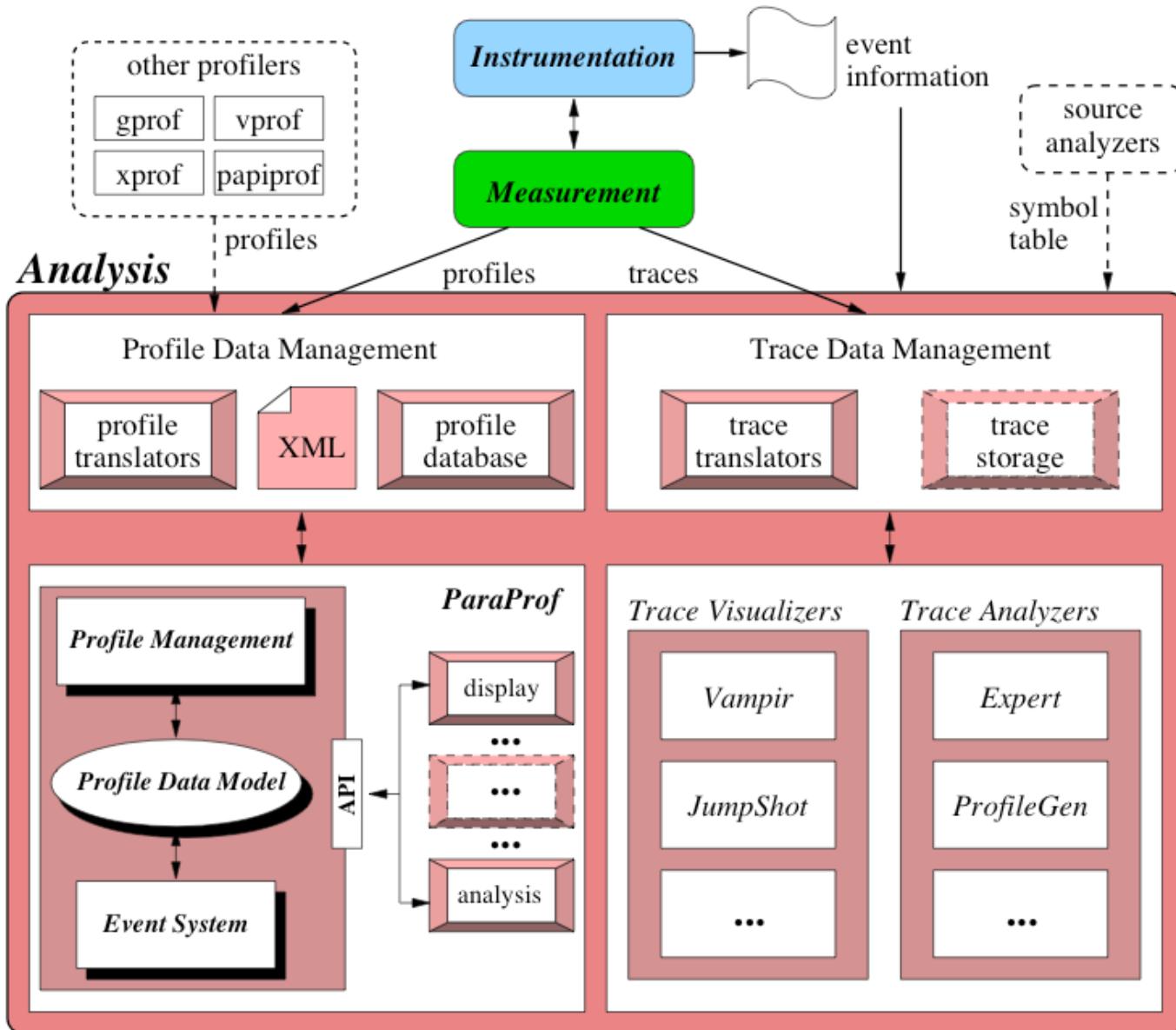
- The rest of the talk will describe what options you can choose for measurement and instrumentation!

TAU Performance System Architecture

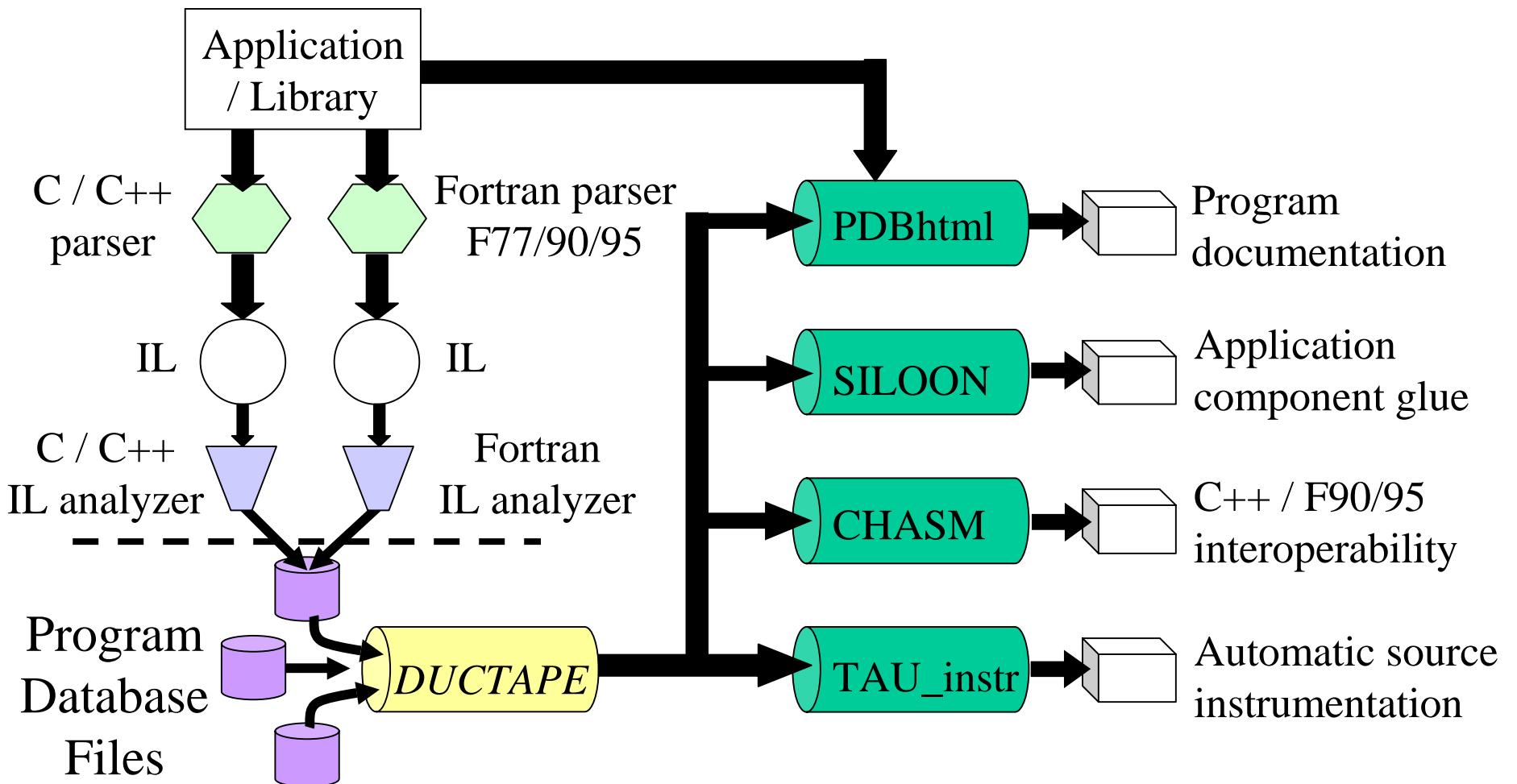




TAU Performance System Architecture



Program Database Toolkit (PDT)





TAU Instrumentation Approach

- Support for standard program events
 - Routines
 - Classes and templates
 - Statement-level blocks
- Support for user-defined events
 - Begin/End events (“user-defined timers”)
 - Atomic events (e.g., size of memory allocated/freed)
 - Selection of event statistics
- Support definition of “semantic” entities for mapping
- Support for event groups
- Instrumentation optimization (eliminate instrumentation in lightweight routines)



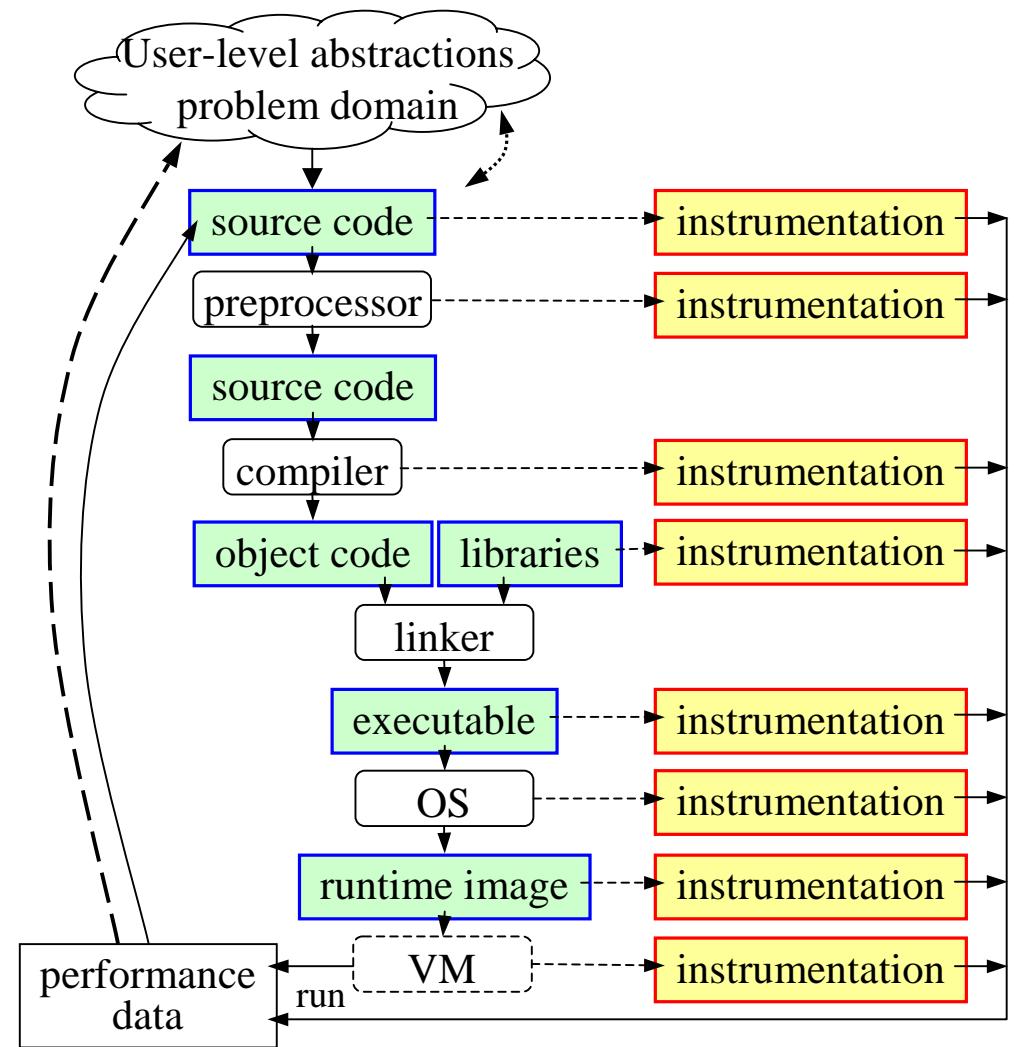
TAU Instrumentation

- Flexible instrumentation mechanisms at multiple levels
 - Source code
 - manual (TAU API, TAU Component API)
 - automatic
 - C, C++, F77/90/95 (Program Database Toolkit (*PDT*))
 - OpenMP (directive rewriting (*Opari*), *POMP spec*)
 - Object code
 - pre-instrumented libraries (e.g., MPI using *PMPI*)
 - statically-linked and dynamically-linked
 - Executable code
 - dynamic instrumentation (pre-execution) (*DynInstAPI*)
 - virtual machine instrumentation (e.g., Java using *JVMPI*)
 - Python interpreter based instrumentation at runtime
 - Proxy Components

Multi-Level Instrumentation and Mapping



- Multiple instrumentation interfaces
- Information sharing
 - Between interfaces
- Event selection
 - Within/between levels
- Mapping
 - Associate performance data with high-level semantic abstractions
- Instrumentation targets measurement API with support for mapping





TAU Measurement Approach

- Portable and scalable parallel profiling solution
 - Multiple profiling types and options
 - Event selection and control (enabling/disabling, throttling)
 - Online profile access and sampling
 - Online performance profile overhead compensation
- Portable and scalable parallel tracing solution
 - Trace translation to Open Trace Format (OTF)
 - Trace streams and hierarchical trace merging
- Robust timing and hardware performance support
- Multiple counters (hardware, user-defined, system)
- Performance measurement for CCA component software



Using TAU

- Configuration
- Instrumentation
 - Manual
 - MPI – Wrapper interposition library
 - PDT- Source rewriting for C,C++, F77/90/95
 - OpenMP – Directive rewriting
 - Component based instrumentation – Proxy components
 - Binary Instrumentation
 - DyninstAPI – Runtime Instrumentation/Rewriting binary
 - Java – Runtime instrumentation
 - Python – Runtime instrumentation
- Measurement
- Performance Analysis



TAU Measurement System Configuration

□ configure [OPTIONS]

{-c++=<CC>, -cc=<cc>}	Specify C++ and C compilers
{-pthread, -sproc}	Use pthread or SGI sproc threads
-openmp	Use OpenMP threads
-jdk=<dir>	Specify Java instrumentation (JDK)
-opari=<dir>	Specify location of O pari OpenMP tool
-papi=<dir>	Specify location of PAPI
-pdt=<dir>	Specify location of PDT
-dyninst=<dir>	Specify location of DynInst Package
-mpi[inc/lib]=<dir>	Specify MPI library instrumentation
-shmem[inc/lib]=<dir>	Specify PSHMEM library instrumentation
-python[inc/lib]=<dir>	Specify Python instrumentation
-tag=<name>	Specify a unique configuration name
-epilog=<dir>	Specify location of EPILOG
-slog2	Build SLOG2/Jumpshot tracing package
-otf=<dir>	Specify location of OTF trace package
-arch=<architecture>	Specify architecture explicitly



TAU Measurement System Configuration

- configure [OPTIONS]
 - TRACE Generate binary TAU traces
 - PROFILE (default) Generate profiles (summary)
 - PROFILECALLPATH Generate call path profiles
 - PROFILEPHASE Generate phase based profiles
 - PROFILEMEMORY Track heap memory for each routine
 - PROFILEHEADROOM Track memory headroom to grow
 - MULTIPLECOUNTERS Use hardware counters + time
 - COMPENSATE Compensate timer overhead
 - CPUTIME Use usertime+system time
 - PAPIWALLCLOCK Use PAPI's wallclock time
 - PAPIVIRTUAL Use PAPI's process virtual time
 - SGITIMERS Use fast IRIX timers
 - LINUXTIMERS Use fast x86 Linux timers



TAU Measurement Configuration – Examples

- `./configure -c++=xlC_r -pthread`
 - Use TAU with xlC_r and pthread library under AIX
 - Enable TAU profiling (default)
- `./configure -TRACE -PROFILE`
 - Enable both TAU profiling and tracing
- `./configure -c++=xlC_r -cc=xlc_r -fortran=ibm64 -papi=/usr/local/packages/papi -pdt=/usr/local/pdtoolkit-3.9 -arch=ibm64 -mpi -MULTIPLECOUNTERS`
 - Use IBM's xlC_r and xlc_r compilers with PAPI, PDT, MPI packages and multiple counters for measurements
- Typically configure multiple measurement libraries
- Each configuration creates a unique <arch>/lib/Makefile.tau-<options> stub makefile that corresponds to the configuration options specified. e.g.,
 - /usr/common/acts/TAU/2.15.5/rs6000/lib/Makefile.tau-mpi-pdt
 - /usr/common/acts/TAU/2.15.5/rs6000/lib/Makefile.tau-mpi-pdt-trace



TAU Measurement Configuration – Examples

```
% cd $(TAUROOTDIR)/rs6000/lib; ls Makefile.*  
Makefile.tau-pdt  
Makefile.tau-mpi-pdt  
Makefile.tau-callpath-mpi-pdt  
Makefile.tau-mpi-pdt-trace  
Makefile.tau-mpi-compensate-pdt  
Makefile.tau-pthread-pdt  
Makefile.tau-papiwallclock-multiplecounters-papivirtual-mpi-papi-pdt  
Makefile.tau-multiplecounters-mpi-papi-pdt-trace  
Makefile.tau-mpi-pdt-epilog-trace  
Makefile.tau-papiwallclock-multiplecounters-papivirtual-papi-pdt-openmp-opari  
...
```

For an MPI+F90 application, you may want to start with:

- Makefile.tau-mpi-pdt
 - Supports MPI instrumentation & PDT for automatic source instrumentation for



Configuration Parameters in Stub Makefiles

- Each TAU stub Makefile resides in <tau>/<arch>/lib directory
- Variables:
 - **TAU_CXX** Specify the C++ compiler used by TAU
 - **TAU_CC, TAU_F90** Specify the C, F90 compilers
 - **TAU_DEFS** Defines used by TAU. Add to CFLAGS
 - **TAU_LDFLAGS** Linker options. Add to LDFLAGS
 - **TAU_INCLUDE** Header files include path. Add to CFLAGS
 - **TAU_LIBS** Statically linked TAU library. Add to LIBS
 - **TAU_SHLIBS** Dynamically linked TAU library
 - **TAU_MPI_LIBS** TAU's MPI wrapper library for C/C++
 - **TAU_MPI_FLIBS** TAU's MPI wrapper library for F90
 - **TAU_FORTRANLIBS** Must be linked in with C++ linker for F90
 - **TAU_CXXLIBS** Must be linked in with F90 linker
 - **TAU_INCLUDE_MEMORY** Use TAU's malloc/free wrapper lib
 - **TAU_DISABLE** TAU's dummy F90 stub library
 - **TAU_COMPILER** Instrument using tau_compiler.sh script
- Each stub makefile encapsulates the parameters that TAU was configured with
- It represents a specific instance of the TAU libraries. TAU scripts use stub makefiles to identify what performance measurements are to be performed.



Using TAU

- **Install TAU**
% configure [options]; make clean install
- **Instrument application manually/automatically**
 - TAU Profiling API
- **Typically modify application makefile**
 - Select TAU's stub makefile, change name of compiler in Makefile
- **Set environment variables**
 - **TAU_MAKEFILE** stub makefile
 - directory where profiles/traces are to be stored
- **Execute application**
% mpirun –np <procs> a.out;
- **Analyze performance data**
 - paraprof, vampir, pprof, paraver ...



TAU's MPI Wrapper Interposition Library

- Uses standard MPI Profiling Interface
 - Provides name shifted interface
 - MPI_Send = PMPI_Send
 - Weak bindings
- Interpose TAU's MPI wrapper library between MPI and TAU
 - -lmpi replaced by -lTauMpi -lpmpi -lmpi
- No change to the source code!
 - Just re-link the application to generate performance data
 - setenv TAU_MAKEFILE <dir>/<arch>/lib/Makefile.tau-mpi - [options]
 - Use tau_cxx.sh, tau_f90.sh and tau_cc.sh as compilers



-PROFILE Configuration Option

- Generates flat profiles (one for each MPI process)
 - It is the default option.
- Uses wallclock time (gettimeofday() sys call)
- Calculates exclusive, inclusive time spent in each timer and number of calls

% pprof

The screenshot shows a terminal window titled "emacs@neutron.cs.uoregon.edu" displaying MPI profiling results. The window title bar includes "Buffers Files Tools Edit Search Mule Help". The main content area shows the following text:

```
Reading Profile files in profile.*  
NODE 0;CONTEXT 0;THREAD 0:  
-----  
%Time Exclusive Inclusive #Call #Subrs Inclusive Name  
msec total msec usec/call  
-----  
100.0 1 3:11.293 1 15 191293269 applu  
99.6 3,667 3:10.463 3 37517 63487925 bcast_inputs  
67.1 491 2:08.326 37200 37200 3450 exchange_1  
44.5 6,461 1:25.159 9300 18600 9157 buts  
41.0 1:18.436 1:18.436 18600 0 4217 MPI_Recv()  
29.5 6,778 56,407 9300 18600 6065 bts  
26.2 50,142 50,142 19204 0 2611 MPI_Send()  
16.2 24,451 31,031 301 602 103096 rhs  
3.9 7,501 7,501 9300 0 807 jacld  
3.4 838 6,594 604 1812 10918 exchange_3  
3.4 6,590 6,590 9300 0 709 jacu  
2.6 4,989 4,989 608 0 8206 MPI_Wait()  
0.2 0.44 400 1 4 400081 init_comm  
0.2 398 399 1 39 399634 MPI_Init()  
0.1 140 247 1 47616 247086 setiv  
0.1 131 131 57252 0 2 exact  
0.1 89 103 1 2 103168 erhs  
0.1 0.966 96 1 2 96458 read_input  
0.0 95 95 9 0 10603 MPI_Bcast()  
0.0 26 44 1 7937 44878 error  
0.0 24 24 608 0 40 MPI_Irecv()  
0.0 15 15 1 5 15630 MPI_Finalize()  
0.0 4 12 1 1700 12335 setbv  
0.0 7 8 3 3 2893 12norm  
0.0 3 3 8 0 491 MPI_Allreduce()  
0.0 1 3 1 6 3874 pintgr  
0.0 1 1 1 0 1007 MPI_Barrier()  
0.0 0.116 0.837 1 4 837 exchange_4  
0.0 0.512 0.512 1 0 512 MPI_Keyval_create()  
0.0 0.121 0.353 1 2 353 exchange_5  
0.0 0.024 0.191 1 2 191 exchange_6  
0.0 0.103 0.103 6 0 17 MPI_Type_contiguous()
```

At the bottom of the window, it says "---- NPB_LU.out (Fundamental)--L8--Top----".



Terminology – Example

- For routine “int main()”:
- Exclusive time
 - 100-20-50-20=10 secs
- Inclusive time
 - 100 secs
- Calls
 - 1 call
- Subrs (no. of child routines called)
 - 3
- Inclusive time/call
 - 100secs

```
int main( )
{ /* takes 100 secs */

    f1(); /* takes 20 secs */
    f2(); /* takes 50 secs */
    f1(); /* takes 20 secs */

    /* other work */
}

/*
Time can be replaced by counts
from PAPI e.g., PAPI_FP_OPS. */
```



-MULTIPLECOUNTERS Configuration Option

- Instead of one metric, profile or trace with more than one metric
 - Set environment variables COUNTER[1-25] to specify the metric
 - % setenv COUNTER1 GET_TIME_OF_DAY
 - % setenv COUNTER2 PAPI_L2_DCM
 - % setenv COUNTER3 PAPI_FP_OPS
 - % setenv COUNTER4 PAPI_NATIVE_<native_event>
 - % setenv COUNTER5 P_WALL_CLOCK_TIME ...
- When used with –TRACE option, the first counter **must** be GET_TIME_OF_DAY
 - % setenv COUNTER1 GET_TIME_OF_DAY
 - Provides a globally synchronized real time clock for tracing
- -multiplecounters appears in the name of the stub Makefile
- Often used with –papi=<dir> to measure hardware performance counters and time
- papi_native and papi_avail are two useful tools



-PROFILECALLPATH Configuration Option

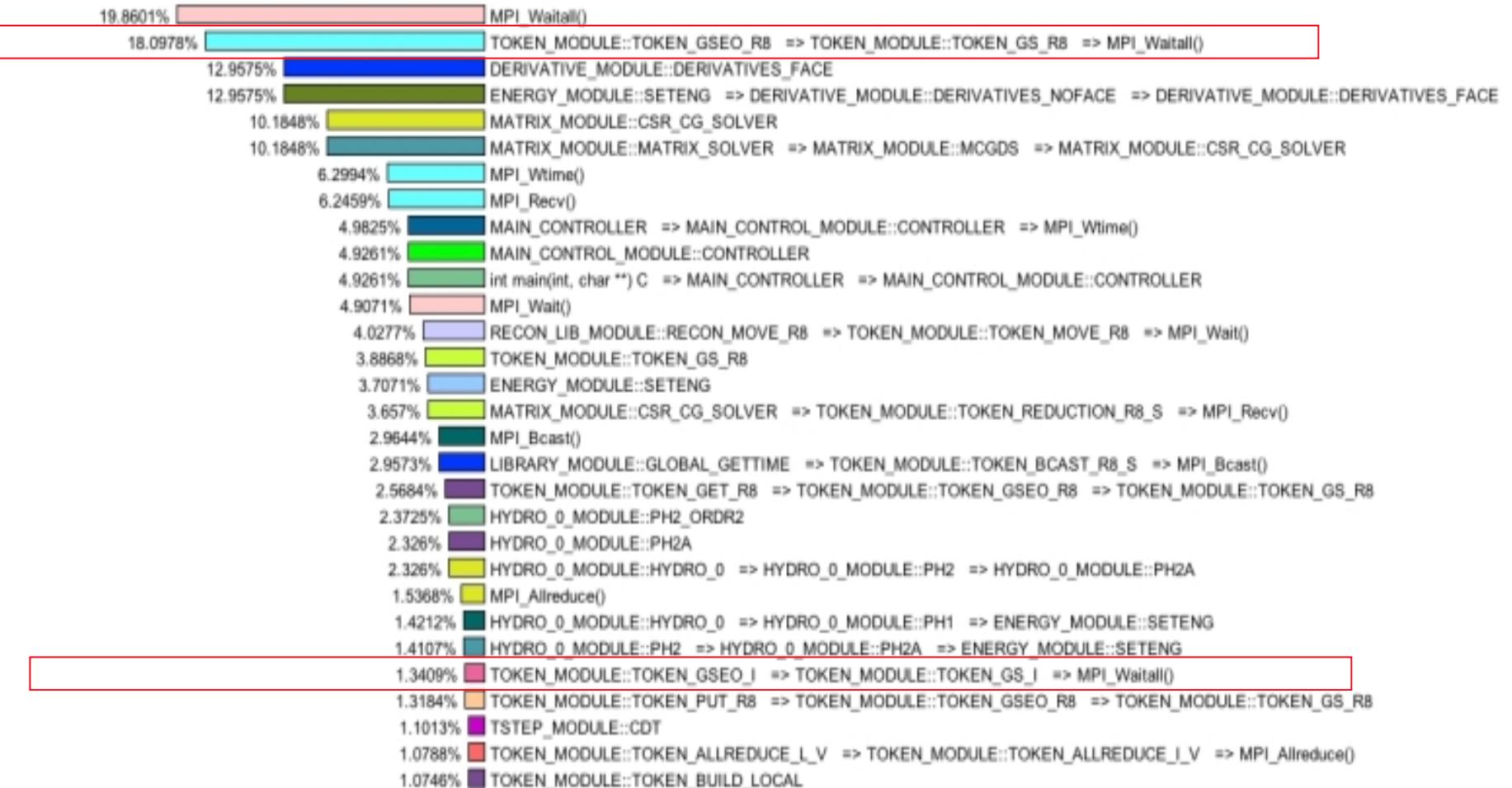
- Generates profiles that show the calling order (edges & nodes in callgraph)
 - A=>B=>C shows the time spent in C when it was called by B and B was called by A
 - Control the depth of callpath using **TAU_CALLPATH_DEPTH** environment variable
 - **-callpath** in the name of the stub Makefile name



-PROFILECALLPATH Configuration Option

Metric Name: Time

Value Type: exclusive





Profile Measurement – Three Flavors

Flat profiles

- Time (or counts) spent in each routine (nodes in callgraph).
- Exclusive/inclusive time, no. of calls, child calls
- E.g.: MPI_Send, foo, ...

Callpath Profiles

- Flat profiles, **plus**
- Sequence of actions that led to poor performance
- Time spent along a calling path (edges in callgraph)
- E.g., “main=> f1 => f2 => MPI_Send” shows the time spent in MPI_Send when called by f2, when f2 is called by f1, when it is called by main. Depth of this callpath = 4 (TAU_CALLPATH_DEPTH environment variable)

Phase based profiles

- Flat profiles, **plus**
- Flat profiles under a phase (nested phases are allowed)
- Default “main” phase has all phases and routines invoked outside phases
- Supports static or dynamic (per-iteration) phases
- E.g., “IO => MPI_Send” is time spent in MPI_Send in IO phase



-DEPTHLIMIT Configuration Option

- Allows users to enable instrumentation at runtime based on the depth of a calling routine on a callstack.
 - Disables instrumentation in all routines a certain depth away from the root in a callgraph
- TAU_DEPTH_LIMIT environment variable specifies depth
 - % setenv TAU_DEPTH_LIMIT 1
enables instrumentation in only “main”
 - % setenv TAU_DEPTH_LIMIT 2
enables instrumentation in main and routines that are directly called by main
- Stub makefile has -depthlimit in its name:
`setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-mpi-depthlimit-pdt`



-COMPENSATE Configuration Option

- Specifies online compensation of performance perturbation
- TAU computes its timer overhead and subtracts it from the profiles
- Works well with time or instructions based metrics
- Does not work with level 1/2 data cache misses



-TRACE Configuration Option

- Generates event-trace logs, rather than summary profiles
- Traces show when and where an event occurred in terms of location and the process that executed it
- Traces from multiple processes are merged:

% tau_treemerge.pl

➤ generates tau.trc and tau.edf as merged trace and event definition file

- TAU traces can be converted to Vampir's OTF/VTF3, Jumpshot SLOG2, Paraver trace formats:

% tau2otf tau.trc tau.edf app.otf

% tau2vtf tau.trc tau.edf app.vpt.gz

% tau2slog2 tau.trc tau.edf -o app.slog2

% tau_convert -paraver tau.trc tau.edf app.prv

- Stub Makefile has **-trace** in its name

% setenv TAU_MAKEFILE <taudir>/<arch>/lib/
Makefile.tau-mpi-pdt-**trace**



-PROFILEPARAM Configuration Option

- Idea: partition performance data for individual functions based on runtime parameters
- Enable by configuring with **-PROFILEPARAM**
- TAU call: **TAU_PROFILE_PARAM1L** (value, “name”)
- Stub makefile has **-param** in its name
- Simple example:

```
void foo(long input) {  
    TAU_PROFILE("foo", "", TAU_DEFAULT);  
    TAU_PROFILE_PARAM1L(input, "input");  
    ... }
```



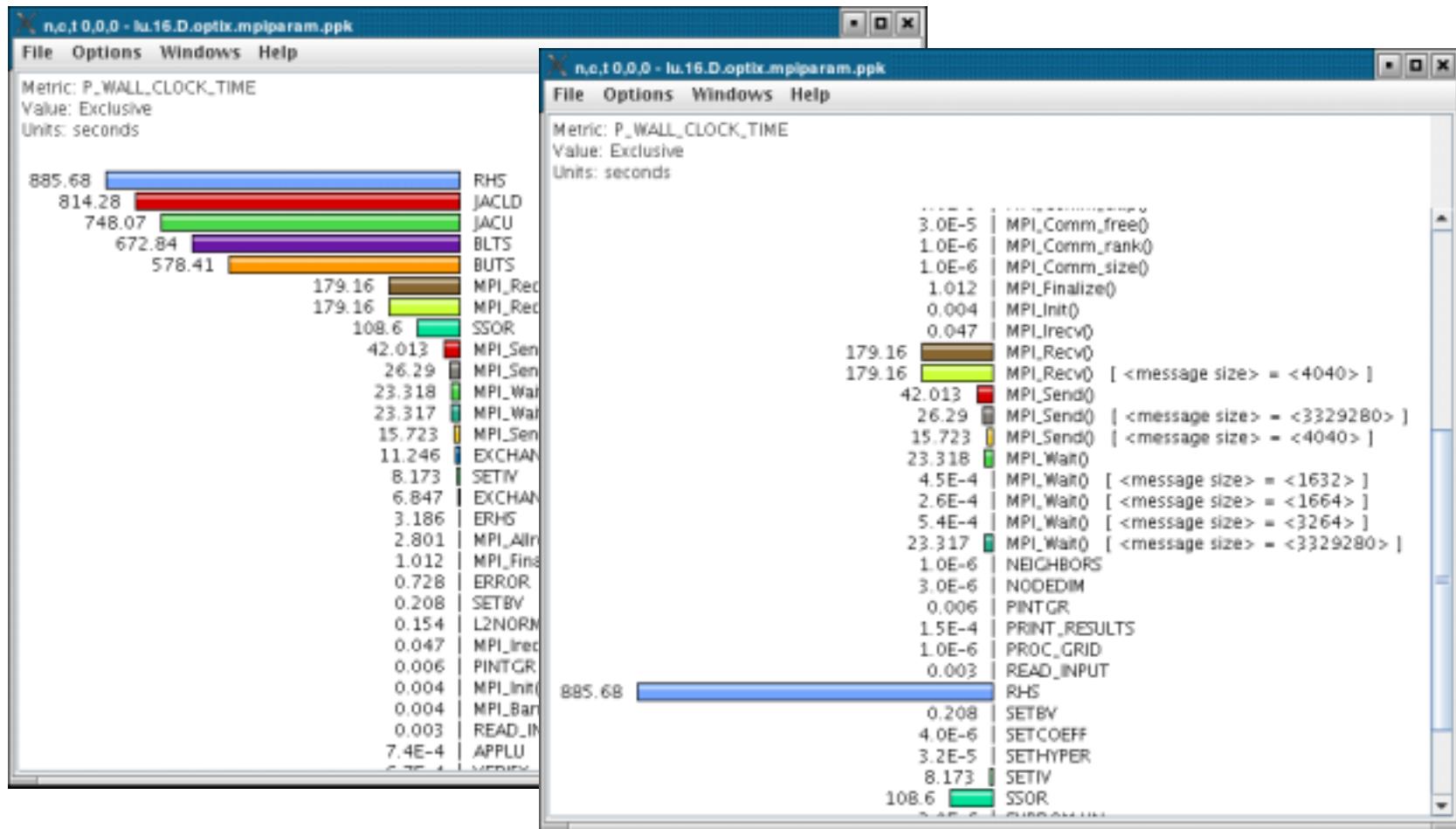
Workload Characterization

- 5 seconds spent in function “`foo`” becomes
 - 2 seconds for “`foo [<input> = <25>]`”
 - 1 seconds for “`foo [<input> = <5>]`”
 - ...
- Currently used in MPI wrapper library
 - Allows for partitioning of time spent in MPI routines based on parameters (message size, message tag, destination node)
 - Can be extrapolated to infer specifics about the MPI subsystem and system as a whole



Workload Characterization

- MPI Results (NAS Parallel Benchmark 3.1, LU class D on 16 processors of SGI Altix)



Workload Characterization

- Two different message sizes (~3.3MB and ~4K)

Thread Statistics: n,c,t,0,0,0 - lu.16.D.optix.mpiparam.ppk

Name	Inclusive ...	Exclusive...	Calls	Child ...
MPI_Comm_free()	0	0	1	0
MPI_Comm_rank()	0	0	1	0
MPI_Comm_size()	0	0	2	0
MPI_Finalize()	1.012	1.012	1	0
MPI_Init()	0.004	0.004	1	0
MPI_Irecv()	0.047	0.047	612	0
MPI_Recv()	179.165	179.165	244,412	0
MPI_Recv0 [<message size> = <4040>]	179.165	179.165	244,412	0
MPI_Send()	42.013	42.013	245,020	0
MPI_Send0 [<message size> = <3329280>]	26.29	26.29	608	0
MPI_Send0 [<message size> = <4040>]	15.723	15.723	244,412	0
MPI_Wait()	23.318	23.318	612	0
MPI_Wait0 [<message size> = <1632>]	0	0	1	0
MPI_Wait0 [<message size> = <1664>]	0	0	1	0
MPI_Wait0 [<message size> = <3264>]	0.001	0.001	2	0
MPI_Wait0 [<message size> = <3329280>]	23.317	23.317	608	0
NEIGHBORS	0	0	1	0
NODEDIM	0	0	1	0
PINTGR	0.008	0.006	1	6
PRINT_RESULTS	0	0	1	0



Memory Profiling in TAU

- Configuration option –**PROFILEMEMORY**
 - Records global heap memory utilization for each function
 - Takes one sample at beginning of each function and associates the sample with **function name**
- Configuration option -**PROFILEHEADROOM**
 - Records headroom (amount of free memory to grow) for each function
 - Takes one sample at beginning of each function
 - Useful for debugging memory usage on IBM BG/L and Cray XT3.
- Independent of instrumentation/measurement options selected
- No need to insert macros/calls in the source code
- User defined atomic events appear in profiles/traces



Memory Profiling in TAU (Atomic events)

Sorted By: number of userEvents

NumSamples	Max	Min	Mean	Std. Dev	Name
252032	2022.7	1181.2	1534.3	410.04	MODULEHYDRO_1D::HYDRO_1D - Heap Memory (KB)
252032	2022.8	1181.7	1534.3	410.04	MODULEINTRFC::INTRFC - Heap Memory (KB)
104559	2023.2	331.13	1526.6	409.54	MODULEEOS3D::EOS3D - Heap Memory (KB)
63008	2022.7	1182	1534.3	410.01	MODULEUPDATE_SOLN::UPDATE_SOLN - Heap Memory (KB)
55545	2023.3	333.07	1514.2	408.31	DBASETREE::DBASENEIGHBORBLOCKLIST - Heap Memory (KB)
51374	2023	1179.4	1497.7	402.53	AMR_PROLONG_GEN_UNK_FUN - Heap Memory (KB)
42120	2022.7	1187.5	1533.5	409.83	ABUNDANCE_RESTRICT - Heap Memory (KB)
41958	2023	346.12	1514.9	408.39	AMR_RESTRICT_UNK_FUN - Heap Memory (KB)
31832	2022.8	1187.4	1534.1	409.91	AMR_RESTRICT_RED - Heap Memory (KB)
31504	2022.7	1181.8	1534.3	410.04	DIFFUSE - Heap Memory (KB)
26042	2023	1179.2	1501.9	403.61	AMR_PROLONG_UNK_FUN - Heap Memory (KB)

Flash2 code profile (-PROFILEMEMORY) on IBM BlueGene/L [MPI rank 0]



Memory Profiling in TAU

- Instrumentation based observation of global heap memory (not per function)
 - call TAU_TRACK_MEMORY()
 - call TAU_TRACK_MEMORY_HEADROOM()
 - Triggers one sample every 10 secs
 - call TAU_TRACK_MEMORY_HERE()
 - call TAU_TRACK_MEMORY_HEADROOM_HERE()
 - Triggers sample at a specific location in source code
 - call TAU_SET_INTERRUPT_INTERVAL(seconds)
 - To set inter-interrupt interval for sampling
 - call TAU_DISABLE_TRACKING_MEMORY()
 - call TAU_DISABLE_TRACKING_MEMORY_HEADROOM()
 - To turn off recording memory utilization
 - call TAU_ENABLE_TRACKING_MEMORY()
 - call TAU_ENABLE_TRACKING_MEMORY_HEADROOM()
 - To re-enable tracking memory utilization



Detecting Memory Leaks in C/C++

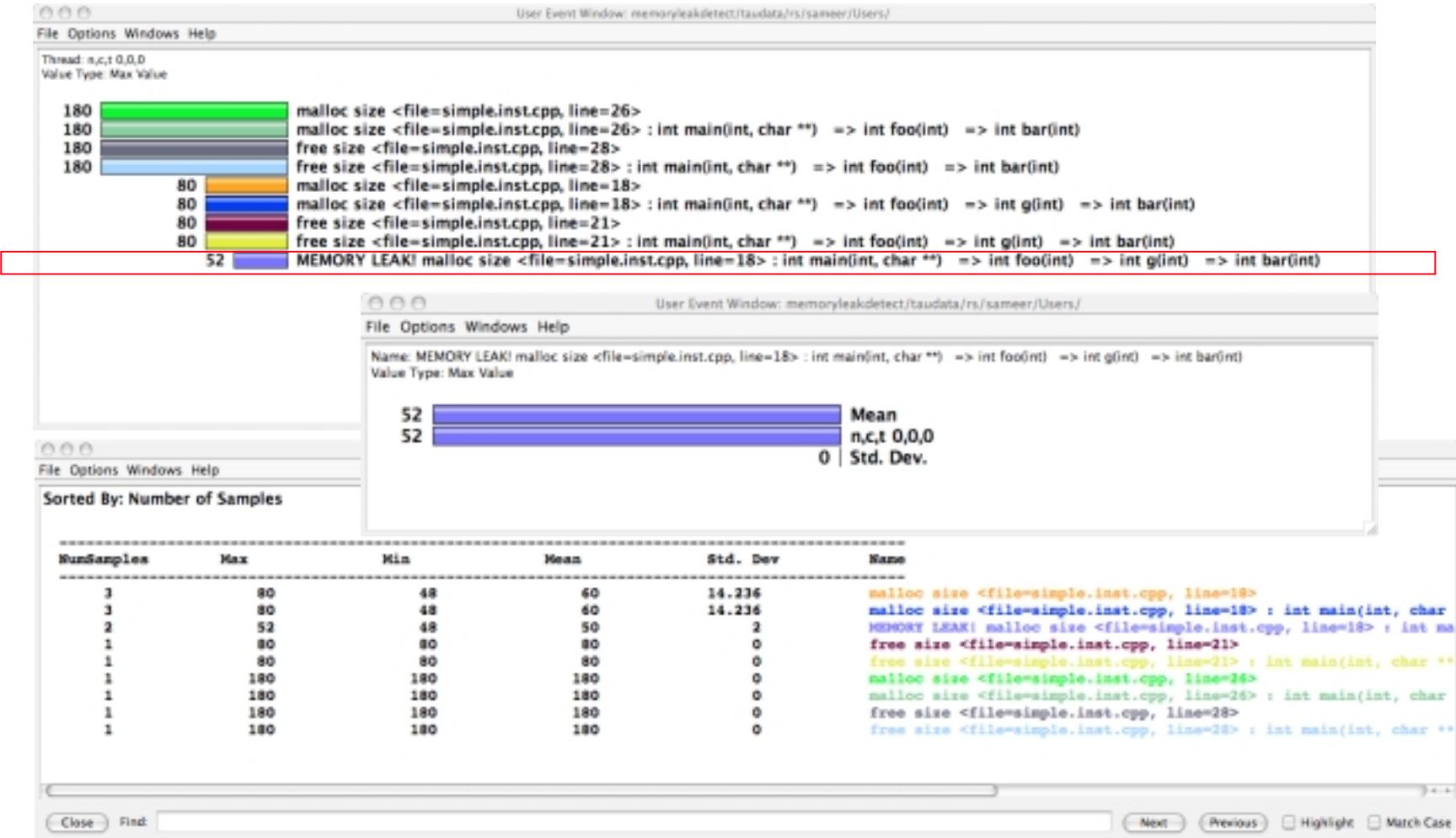
- TAU wrapper library for malloc/realloc/free
- During instrumentation, specify
 - optDetectMemoryLeaks option to TAU_COMPILER
 - % setenv TAU_OPTIONS '-optVerbose -optDetectMemoryLeaks'
 - % setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-mpi-pdt...
 - % tau_cxx.sh foo.cpp ...
- Tracks each memory allocation/de-allocation in parsed files
- Correlates each memory event with the executing callstack
- At the end of execution, TAU detects memory leaks
- TAU reports leaks based on allocations and the executing callstack
- Set **TAU_CALLPATH_DEPTH** environment variable to limit callpath data
 - default is 2
- Future work
 - Support for C++ new/delete planned
 - Support for Fortran 90/95 allocate/deallocate planned



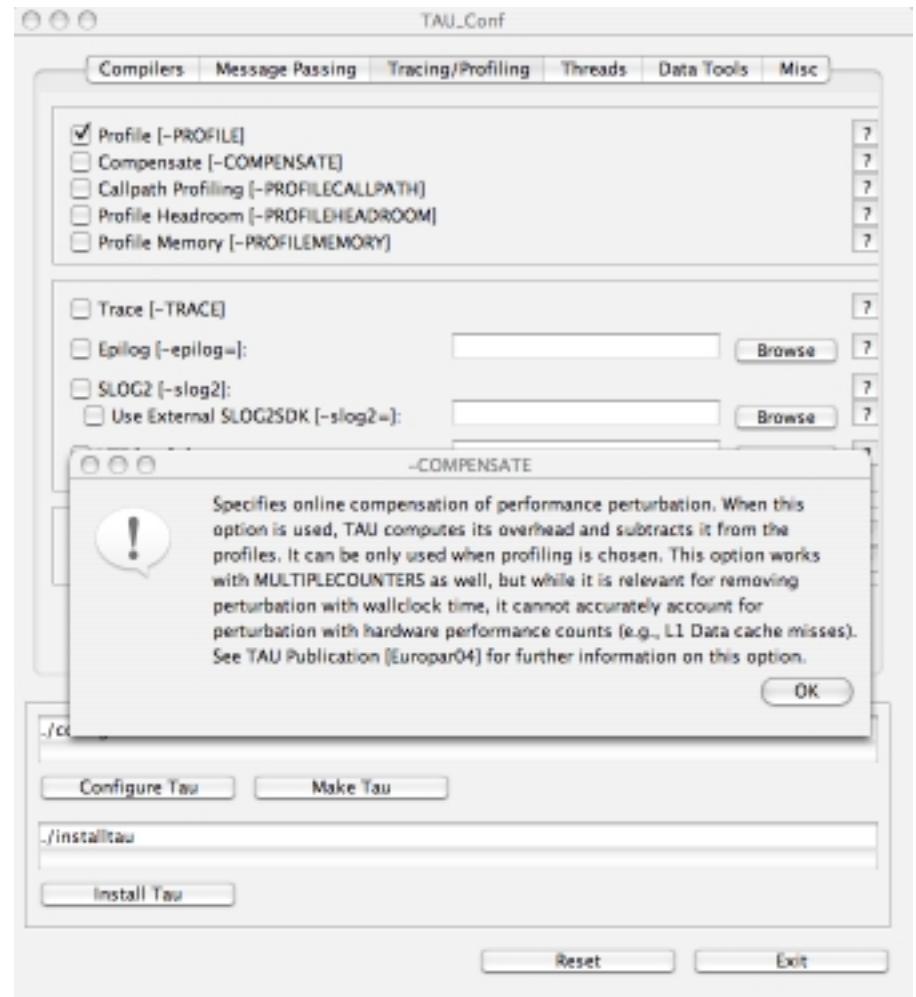
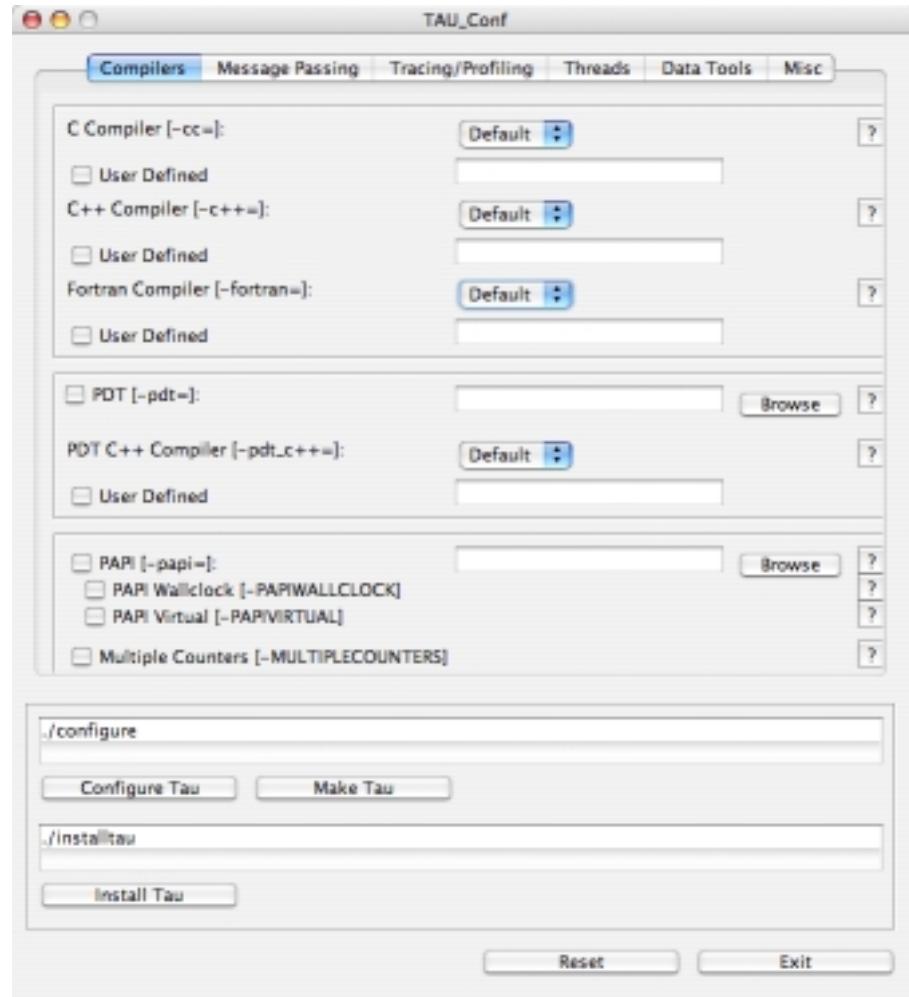
Detecting Memory Leaks in C/C++

```
include /opt/tau/rs6000/lib/Makefile.tau-mpi-pdt
MYOPTS = -optVerbose -optDetectMemoryLeaks
CC= $(TAU_COMPILER) $(MYOPTS) $(TAU_CXX)
LIBS = -lm
OBJS = f1.o f2.o ...
TARGET= a.out
TARGET: $(OBJS)
        $(F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.c.o:
        $(CC) $(CFLAGS) -c $< -o $@
```

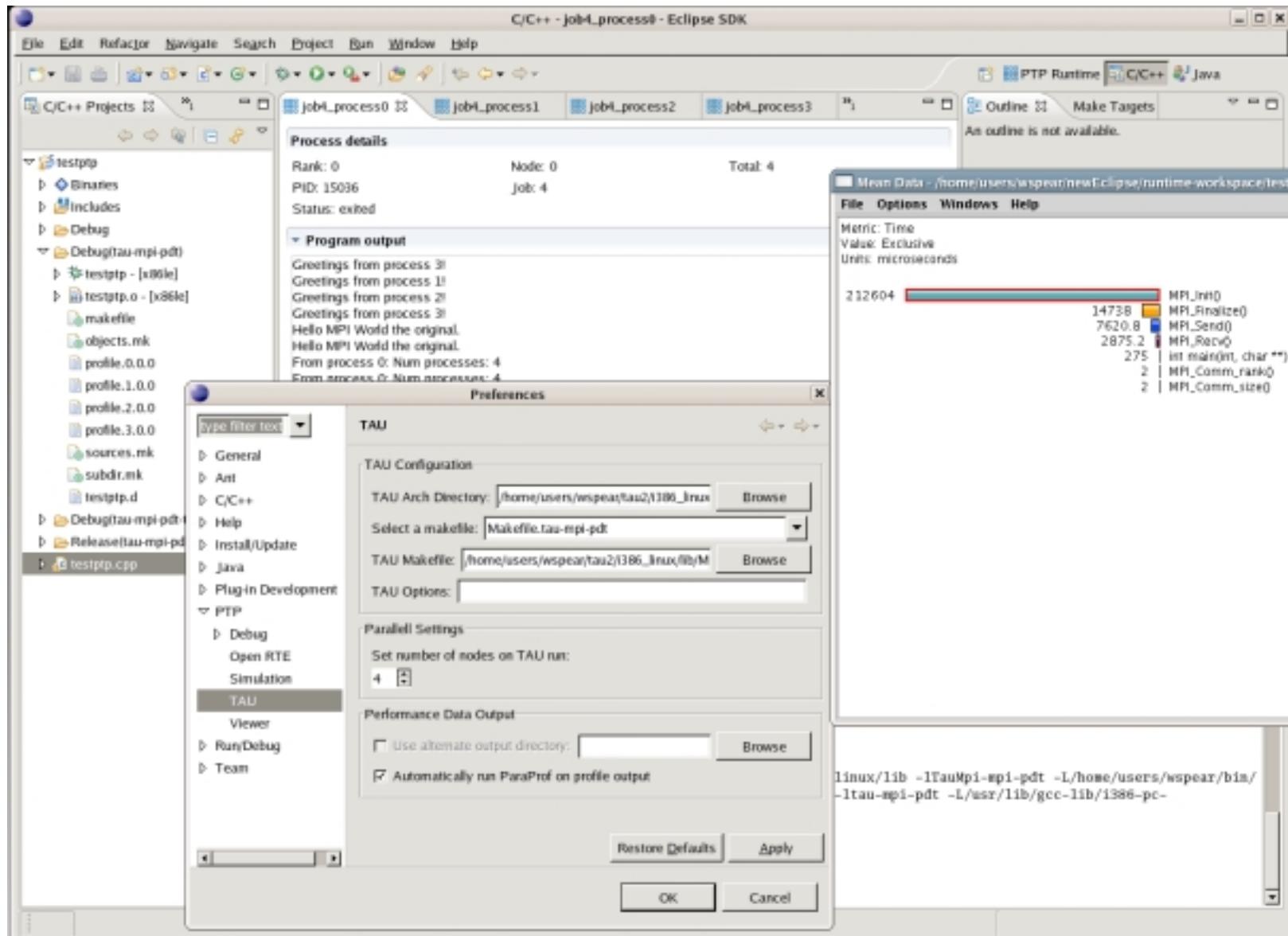
Memory Leak Detection



TAU_SETUP: A GUI for Installing TAU



TAU integration in Eclipse PTP IDE



TAU Manual Instrumentation API for C/C++



- Initialization and runtime configuration
 - TAU_PROFILE_INIT(argc, argv);
TAU_PROFILE_SET_NODE(myNode);
TAU_PROFILE_SET_CONTEXT(myContext);
TAU_PROFILE_EXIT(message);
TAU_REGISTER_THREAD();
- Function and class methods for C++ only:
 - TAU_PROFILE(name, type, group);
 - TAU_PROFILE (name, type, group);
- Template
 - TAU_TYPE_STRING(variable, type);
TAU_PROFILE(name, type, group);
CT (variable);
- User-defined timing
 - TAU_PROFILE_TIMER(timer, name, type, group);
TAU_PROFILE_START(timer);
TAU_PROFILE_STOP(timer);



TAU Measurement API (continued)

- Defining application phases
 - TAU_PHASE_CREATE_STATIC(var, name, type, group);
 - TAU_PHASE_CREATE_DYNAMIC(var, name, type, group);
 - TAU_PHASE_START(var)
 - TAU_PHASE_STOP (var)
- User-defined events
 - TAU_REGISTER_EVENT(variable, event_name);
 - TAU_EVENT(variable, value);
 - TAU_PROFILE_STMT(statement);
- Heap Memory Tracking:
 - TAU_TRACK_MEMORY();
 - TAU_TRACK_MEMORY_HEADROOM();
 - TAU_SET_INTERRUPT_INTERVAL(seconds);
 - TAU_DISABLE_TRACKING_MEMORY[_HEADROOM]();
 - TAU_ENABLE_TRACKING_MEMORY[_HEADROOM]();



Manual Instrumentation – C++ Example

```
#include <TAU.h>
int main(int argc, char **argv)
{
    TAU_PROFILE("int main(int, char **)", " ", TAU_DEFAULT);
    TAU_PROFILE_INIT(argc, argv);
    TAU_PROFILE_SET_NODE(0); /* for sequential programs */
    foo();
    return 0;
}
int foo(void)
{
    TAU_PROFILE("int foo(void)", " ", TAU_DEFAULT); // measures entire foo()
    TAU_PROFILE_TIMER(t, "foo(): for loop", "[23:45 file.cpp]", TAU_USER);
    TAU_PROFILE_START(t);
    for(int i = 0; i < N ; i++){
        work(i);
    }
    TAU_PROFILE_STOP(t);
    // other statements in foo ...
}
```



Manual Instrumentation – F90 Example

```
cc34567 Cubes program - comment line

PROGRAM SUM_OF_CUBES
    integer profiler(2)
    save profiler
    INTEGER :: H, T, U
    call TAU_PROFILE_INIT()
    call TAU_PROFILE_TIMER(profiler, 'PROGRAM SUM_OF_CUBES')
    call TAU_PROFILE_START(profiler)
    call TAU_PROFILE_SET_NODE(0)

! This program prints all 3-digit numbers that equal the sum of the cubes of
their digits.

    DO H = 1, 9
        DO T = 0, 9
            DO U = 0, 9
                IF (100*H + 10*T + U == H**3 + T**3 + U**3) THEN
                    PRINT "(3I1)", H, T, U
                ENDIF
            END DO
        END DO
    END DO
    call TAU_PROFILE_STOP(profiler)
END PROGRAM SUM_OF_CUBES
```



TAU Timers and Phases

Static timer

- Shows time spent in all invocations of a routine (foo)
- E.g., “foo()” 100 secs, 100 calls

Dynamic timer

- Shows time spent in each invocation of a routine
- E.g., “foo() 3” 4.5 secs, “foo 10” 2 secs (invocations 3 and 10 respectively)

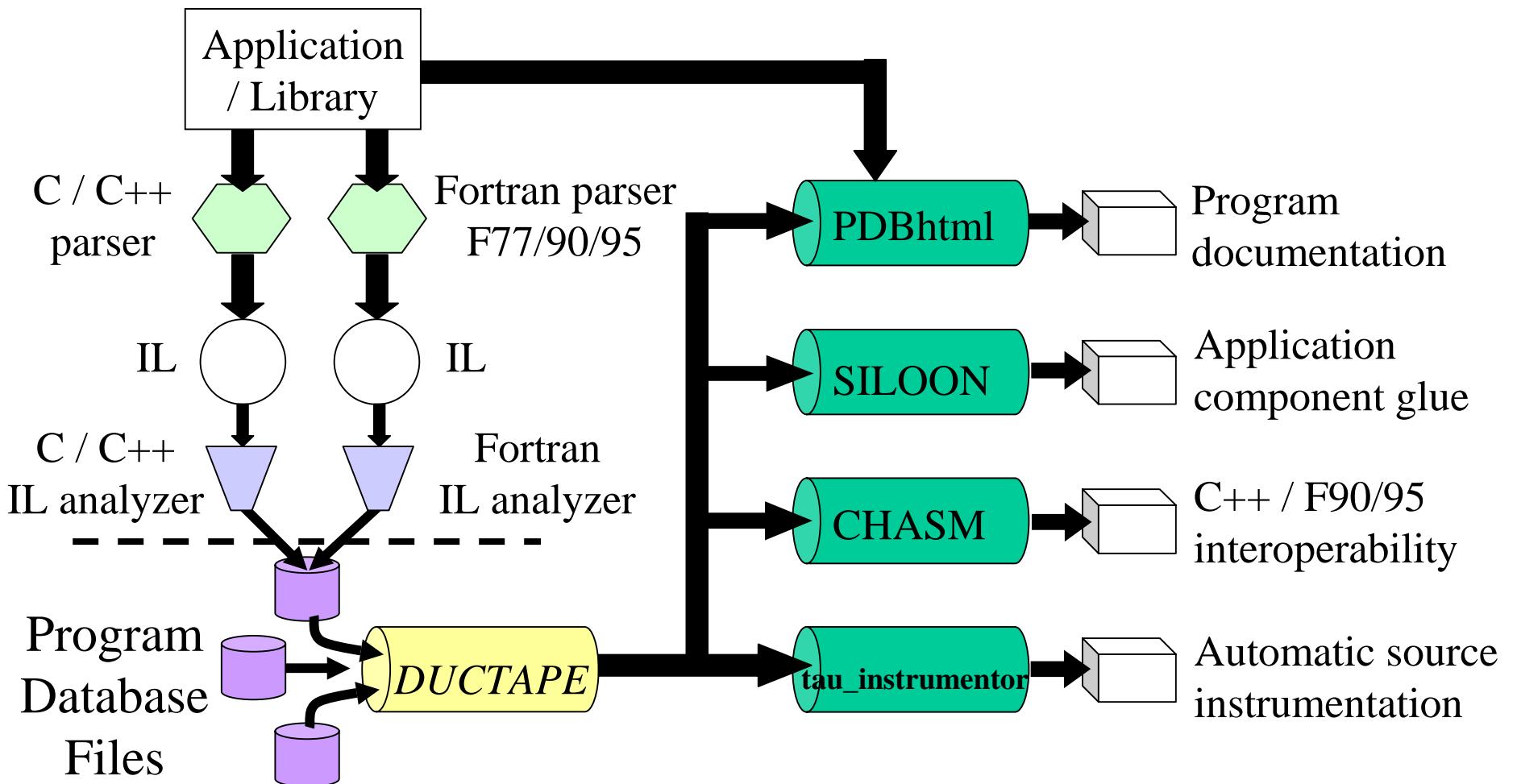
Static phase

- Shows time spent in all routines called (directly/indirectly) by a given routine (foo)
- E.g., “foo() => MPI_Send()” 100 secs, 10 calls shows that a total of 100 secs were spent in MPI_Send() when it was called by foo.

Dynamic phase

- Shows time spent in all routines called by a given invocation of a routine.
- E.g., “foo() 4 => MPI_Send()” 12 secs, shows that 12 secs were spent in MPI_Send when it was called by the 4th invocation of foo.

Program Database Toolkit (PDT)



Using TAU



- Install TAU
 - Configuration
 - Measurement library creation
- Instrument application
 - Manual or automatic source instrumentation
 - Instrumented library (e.g., MPI – wrapper interposition library)
- Create performance experiments
 - Integrate with application build environment
 - Set experiment variables
-
- Execute application
- Analyze performance



Integration with Application Build Environment

- Try to minimize impact on user's application build procedures
- Handle process of parsing, instrumentation, compilation, linking
- Dealing with Makefiles
 - Minimal change to application Makefile
 - Avoid changing compilation rules in application Makefile
 - No explicit inclusion of rules for process stages
- Some applications do not use Makefiles
 - Facilitate integration in whatever procedures used
- Two techniques:
 - TAU shell scripts (`tau_<compiler>.sh`)
 - Invokes all PDT parser, TAU instrumenter, and compiler
 - TAU_COMPILER



Using Program Database Toolkit (PDT)

1. Parse the Program to create foo.pdb:

```
% cxxparse foo.cpp -I/usr/local/mydir -DMYFLAGS ...
```

or

```
% cparsse foo.c -I/usr/local/mydir -DMYFLAGS ...
```

or

```
% f95parse foo.f90 -I/usr/local/mydir ...
```

```
% f95parse *.f -omerged.pdb -I/usr/local/mydir -R free
```

2. Instrument the program:

```
% tau_instrumentor foo.pdb foo.f90 -o foo.inst.f90  
-f select.tau
```

3. Compile the instrumented program:

```
% ifort foo.inst.f90 -c -I/usr/local/mpi/include -o foo.o
```



Tau_[cxx,cc,f90].sh – Improves Integration in Makefiles

```
# set TAU_MAKEFILE and TAU_OPTIONS env vars
CC = tau_cc.sh
F90 = tau_f90.sh
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o ... fn.o

app: $(OBJS)
    $(F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)

.c.o:
    $(CC) $(CFLAGS) -c $<
.f90.o:
    $(F90) $(FFLAGS) -c $<
```



AutoInstrumentation using TAU_COMPILER

- \$(TAU_COMPILER) stub Makefile variable
- Invokes PDT parser, TAU instrumentor, compiler through **`tau_compiler.sh`** shell script
- Requires minimal changes to application Makefile
 - Compilation rules are not changed
 - User adds \$(TAU_COMPILER) before compiler name
 - F90=mpxlf90
Changes to
F90= **\$(TAU_COMPILER)** mpxlf90
- Passes options from TAU stub Makefile to the four compilation stages
- Use **`tau_cxx.sh`, `tau_cc.sh`, `tau_f90.sh`** scripts **OR** **\$(TAU_COMPILER)**
- Uses original compilation command if an error occurs



Automatic Instrumentation

- We now provide compiler wrapper scripts
 - Simply replace `mpxlf90` with `tau_f90.sh`
 - Automatically instruments Fortran source code, links with TAU MPI Wrapper libraries.
- Use `tau_cc.sh` and `tau_cxx.sh` for C/C++

Before

```
CXX = mpCC
F90 = mpxlf90_r
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o ... fn.o

app: $(OBJS)
    $(CXX) $(LDFLAGS) $(OBJS) -o $@
    $(LIBS)

.cpp.o:
    $(CC) $(CFLAGS) -c $<
```

After

```
CXX = tau_cxx.sh
F90 = tau_f90.sh
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o ... fn.o

app: $(OBJS)
    $(CXX) $(LDFLAGS) $(OBJS) -o $@
    $(LIBS)

.cpp.o:
    $(CC) $(CFLAGS) -c $<
```



TAU_COMPILER – Improving Integration in Makefiles

```
include /usr/tau-2.15.5/rs6000/lib/Makefile.tau-mpi-pdt
CXX = $(TAU_COMPILER) mpCC_r
F90 = $(TAU_COMPILER) mpxlf90_r
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o ... fn.o

app: $(OBJS)
    $(CXX) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.cpp.o:
    $(CXX) $(CFLAGS) -c $<
```



TAU_COMPILER Commandline Options

- See `<taudir>/<arch>/bin/tau_compiler.sh -help`

- Compilation:

```
% mpixlf90 -c foo.f90
```

Changes to

```
% f95parse foo.f90 $(OPT1)
```

```
% tau_instrumentor foo.pdb foo.f90 -o foo.inst.f90 $(OPT2)
```

```
% mpixlf90 -c foo.f90 $(OPT3)
```

- Linking:

```
% mpixlf90 foo.o bar.o -o app
```

Changes to

```
% mpixlf90 foo.o bar.o -o app $(OPT4)
```

- Where options OPT[1-4] default values may be overridden by the user:

```
F90 = $(TAU_COMPILER) $(MYOPTIONS) mpixlf90
```



TAU_COMPILER Options

- Optional parameters for \$(TAU_COMPILER): [tau_compiler.sh –help]
 - optVerbose Turn on verbose debugging messages
 - optDetectMemoryLeaks Turn on debugging memory allocations/de-allocations to track leaks
 - optPdtGnuFortranParser Use gfparse (GNU) instead of f95parse (Cleanscape) for parsing Fortran source code
 - optKeepFiles Does not remove intermediate .pdb and .inst.* files
 - optPreProcess Preprocess Fortran sources before instrumentation
 - optTauSelectFile="" Specify selective instrumentation file for tau_instrumentor
 - optLinking="" Options passed to the linker. Typically \$(TAU_MPI_FLIBS) \$(TAU_LIBS) \$(TAU_CXXLIBS)
 - optCompile="" Options passed to the compiler. Typically \$(TAU_MPI_INCLUDE) \$(TAU_INCLUDE) \$(TAU_DEFS)
 - optPdtF95Opts="" Add options for Fortran parser in PDT (f95parse/gfparse)
 - optPdtF95Reset="" Reset options for Fortran parser in PDT (f95parse/gfparse)
 - optPdtCOpts="" Options for C parser in PDT (cparse). Typically \$(TAU_MPI_INCLUDE) \$(TAU_INCLUDE) \$(TAU_DEFS)
 - optPdtCxxOpts="" Options for C++ parser in PDT (cxxparse). Typically \$(TAU_MPI_INCLUDE) \$(TAU_INCLUDE) \$(TAU_DEFS)
- ...



Overriding Default Options:*TAU_COMPILER*

```
include $(TAUROOTDIR)/rs6000/lib/
          Makefile.tau-mpi-pdt-trace
# Fortran .f files in free format need the -R free option for parsing
# Are there any preprocessor directives in the Fortran source?
MYOPTIONS= -optVerbose -optPreProcess -optPdtF950pts=''-R free''
F90 = $(TAU_COMPILER) $(MYOPTIONS) ifort
OBJS = f1.o f2.o f3.o ...
LIBS = -Lappdir -lapplib1 -lapplib2 ...

app: $(OBJS)
      $(F90) $(OBJS) -o app $(LIBS)
.f.o:
      $(F90) -c $<
```



Overriding Default Options: TAU_COMPILER

```
% cat Makefile
F90 = tau_f90.sh
OBJS = f1.o f2.o f3.o ...
LIBS = -Lappdir -lapplib1 -lapplib2 ...

app: $(OBJS)
    $(F90) $(OBJS) -o app $(LIBS)
.f90.o:
    $(F90) -c $<
% setenv TAU_OPTIONS '-optVerbose -optTauSelectFile=select.tau
                     -optKeepFiles'
% setenv TAU_MAKEFILE <taudir>/x86_64/lib/Makefile.tau-mpi-pdt
```



Optimization of Program Instrumentation

- Need to eliminate instrumentation in frequently executing lightweight routines
- Throttling of events at runtime:

```
% setenv TAU_THROTTLE 1
```

Turns off instrumentation in routines that execute over 10000 times

(TAU_THROTTLE_NUMCALLS) and take less than 10 microseconds of inclusive time per call (TAU_THROTTLE_PERCALL)

- Selective instrumentation file to filter events

```
% tau_instrumentor [options] -f <file> OR
```

```
% setenv TAU_OPTIONS '-optTauSelectFile=tau.txt'
```

- Compensation of local instrumentation overhead

```
% configure -COMPENSATE
```



Selective Instrumentation File

- Specify a list of routines to exclude or include (case sensitive)
- # is a wildcard in a routine name. It cannot appear in the first column.

BEGIN_EXCLUDE_LIST

Foo

Bar

D#EMM

END_EXCLUDE_LIST

- Specify a list of routines to include for instrumentation

BEGIN_INCLUDE_LIST

int main(int, char **)

F1

F3

END_LIST_LIST

- Specify either an include list or an exclude list!



Selective Instrumentation File

- Optionally specify a list of files to exclude or include (case sensitive)
- * and ? may be used as wildcard characters in a file name

BEGIN_FILE_EXCLUDE_LIST

f*.f90

Foo?.cpp

END_EXCLUDE_LIST

- Specify a list of routines to include for instrumentation

BEGIN_FILE_INCLUDE_LIST

main.cpp

foo.f90

END_INCLUDE_LIST_LIST



Selective Instrumentation File

- User instrumentation commands are placed in INSTRUMENT section
- ? and * used as wildcard characters for file name, # for routine name
- \ as escape character for quotes
- Routine entry/exit, arbitrary code insertion
- Outer-loop level instrumentation

```
BEGIN_INSTRUMENT_SECTION
loops file="foo.f90" routine="matrix#"
      file="foo.f90" line = 123 code = "  print *, \" Inside foo\""
      exit routine = "int foo()" code = "cout <<\"Exiting foo\"<<endl;"
END_INSTRUMENT_SECTION
```



Instrumentation Specification

```
% tau_instrumentor

Usage : tau_instrumentor < pdbfile > < sourcefile > [ -o < outputfile > ] [ -noinline ]
[ -g groupname ] [ -i headerfile ] [ -c | -c++ | -fortran ] [ -f < instr_req_file > ]

For selective instrumentation, use -f option
% tau_instrumentor foo.pdb foo.cpp -o foo.inst.cpp -f selective.dat
% cat selective.dat

# Selective instrumentation: Specify an exclude/include list of routines/files.

BEGIN_EXCLUDE_LIST
void quicksort(int *, int, int)
void sort_5elements(int *)
void interchange(int *, int *)
END_EXCLUDE_LIST

BEGIN_FILE_INCLUDE_LIST
Main.cpp
Foo?.c
*.C
END_FILE_INCLUDE_LIST

# Instruments routines in Main.cpp, Foo?.c and *.C files only
# Use BEGIN_[FILE]_INCLUDE_LIST with END_[FILE]_INCLUDE_LIST
```



Automatic Outer Loop Level Instrumentation

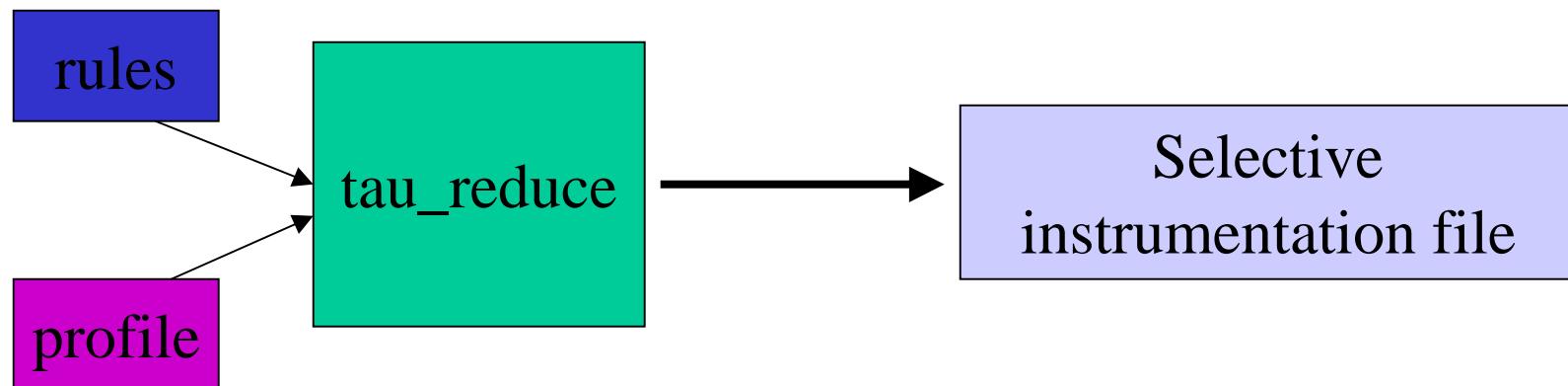
```
BEGIN_INSTRUMENT_SECTION
loops file="loop_test.cpp" routine="multiply"
# it also understands # as the wildcard in routine name
# and * and ? wildcards in file name.
# You can also specify the full
# name of the routine as is found in profile files.
#loops file="loop_test.cpp" routine="double multiply#"
END_INSTRUMENT_SECTION

% pprof
NODE 0;CONTEXT 0;THREAD 0:
-----
%Time      Exclusive      Inclusive      #Call      #Subrs      Inclusive Name
               msec       total msec
                                         usec/call
-----
100.0          0.12        25,162           1           1  25162827 int main(int, char **)
100.0          0.175       25,162           1           4  25162707 double multiply()
 90.5         22,778       22,778           1           0  22778959 Loop: double multiply()[
file = <loop_test.cpp> line,col = <23,3> to <30,3> ]
   9.3          2,345       2,345           1           0  2345823 Loop: double multiply()[
file = <loop_test.cpp> line,col = <38,3> to <46,7> ]
   0.1            33          33           1           0   33964 Loop: double
multiply() [ file = <loop_test.cpp> line,col = <16,10> to <21,12> ]
```

TAU_REDUCE



- Reads profile files and rules
- Creates selective instrumentation file
 - Specifies which routines should be excluded from instrumentation





Optimizing Instrumentation Overhead: Rules

- #Exclude all events that are members of TAU_USER
#and use less than 1000 microseconds
TAU_USER:usec < 1000
- #Exclude all events that have less than 100
#microseconds and are called only once
usec < 1000 & numcalls = 1
- #Exclude all events that have less than 1000 usecs per
#call OR have a (total inclusive) percent less than 5
usecs/call < 1000
percent < 5
- Scientific notation can be used
 - **usec>1000 & numcalls>400000 & usecs/call<30 & percent>25**
- Usage:
% pprof -d > pprof.dat
% tau_reduce -f pprof.dat -r rules.txt -o select.tau

Instrumentation of OpenMP Constructs



- OpenMP Pragma And Region Instrumentor
- Source-to-Source translator to insert **POMP** calls around OpenMP constructs and API functions
- Done: Supports
 - Fortran77 and Fortran90, OpenMP 2.0
 - C and C++, OpenMP 1.0
 - POMP Extensions
 - EPILOG and TAU POMP implementations
 - Preserves source code information (**#line line file**)
- Work in Progress:
Investigating standardization through OpenMP Forum
- KOJAK Project website <http://icl.cs.utk.edu/kojak>



Example: !\$OMP PARALLEL DO Instrumentation

```
call pomp_parallel_fork(d)
!$OMP PARALLEL other-clauses...
    call pomp_parallel_begin(d)
    call pomp_do_enter(d)
    !$OMP DO schedule-clauses, ordered-clauses,
              lastprivate-clauses
        do loop
    !$OMP END DO NOWAIT
    call pomp_barrier_enter(d)
    !$OMP BARRIER
    call pomp_barrier_exit(d)
    call pomp_do_exit(d)
    call pomp_parallel_end(d)
!$OMP END PARALLEL DO
call pomp_parallel_join(d)
```



Using Opari with TAU

Step I: Configure KOJAK/opari [Download from <http://www.fz-juelich.de/zam/kojak/>]

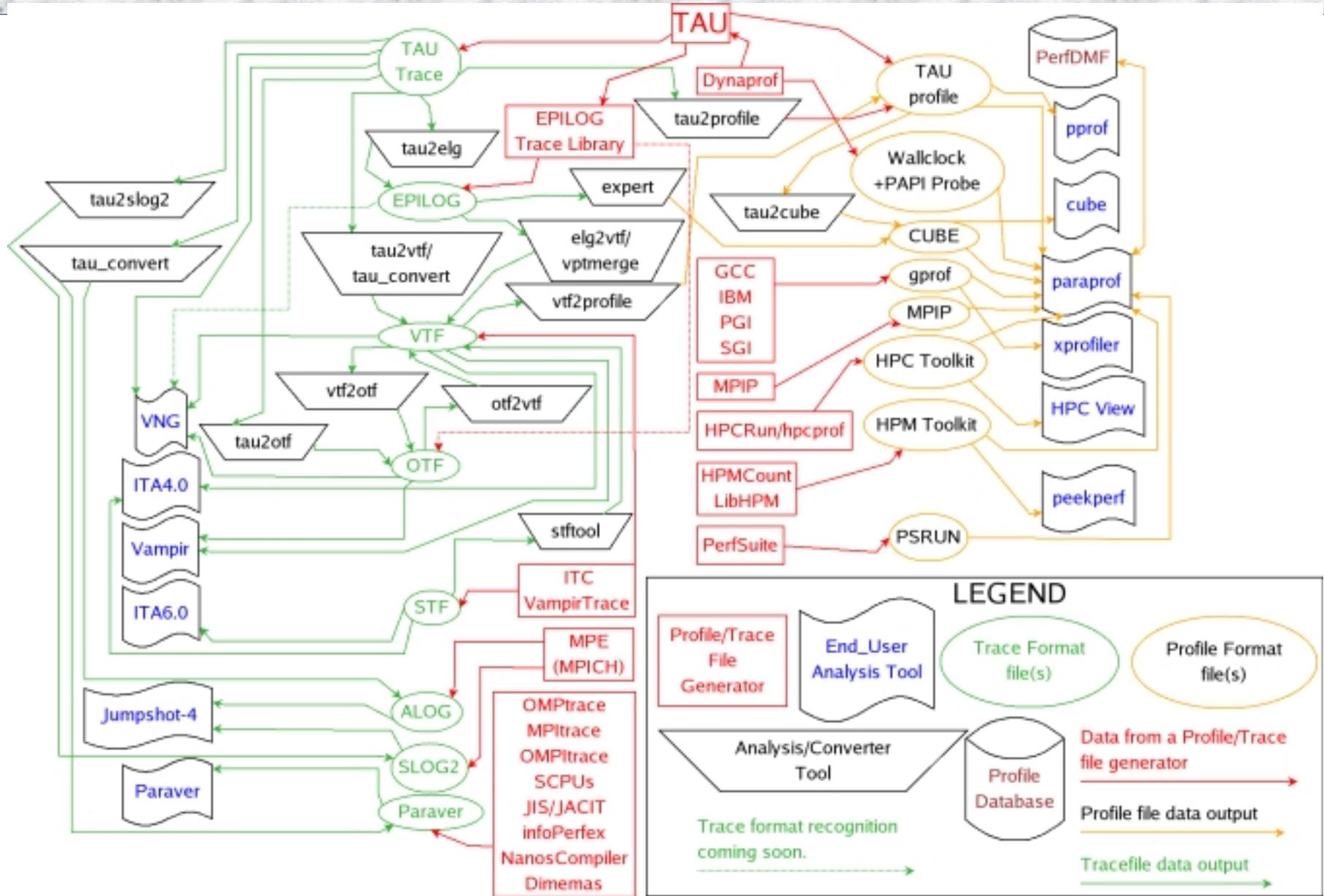
```
% cd kojak-2.1.1; cp mf/Makefile.defs.ibm Makefile.defs;  
edit Makefile  
% make
```

Builds opari

Step II: Configure TAU with Opari (used here with MPI and PDT)

```
% configure -opari=/usr/contrib/TAU/kojak-2.1.1/opari  
-mpiinc=/usr/lpp/ppe.poe/include  
-mpilib=/usr/lpp/ppe.poe/lib  
-pdt=/usr/contrib/TAU/pdtoolkit-3.9  
% make clean; make install  
% setenv TAU_MAKEFILE /tau/<arch>/lib/Makefile.tau-...opari-...  
% tau_cxx.sh -c foo.cpp  
% tau_cxx.sh -c bar.f90  
% tau_cxx.sh *.o -o app
```

Building Bridges to Other Tools: TAU



Advances in TAU Performance Analysis



- Enhanced parallel profile analysis (*ParaProf*)
 - Callpath analysis integration in ParaProf
 - Event callgraph view
- *Performance Data Management Framework (PerfDMF)*
 - First release of prototype
- Integration with *Vampir Next Generation (VNG)*
 - Online trace analysis
- 3D Performance visualization
- Component performance modeling and QoS

ParaProf – Manager Window



performance database

metadata

ParaProf Manager

File Options Help

Applications

- Standard Applications
 - WRF
 - Scaling
 - wrf512.ppk/ppk/sameer/Users/Time
 - DB (jdbc:postgresql://www.paratools.com:5432/paratool_perfmdf)
 - AORSA2D
 - Basic run-time profiling for Socorro
 - CAM3.2.4
 - T31.testTau
 - cam32x1papi.ppk/ppk/Desktop/hammw/Users/GET_TIME_OF_DAY
 - PAPLL1_DCM
 - GAMESS
 - Gyro
 - gyro.B1-std
 - Heap memory management for Socorro
 - HYCOM
 - hydroshock
 - LAMMPS
 - MFIIX
 - mpiP data
 - PNEO
 - POP
 - PTURBO
 - S3D
 - SHAMRC

Field	Value
Name	T31.testTau
Application ID	26
Experiment ID	52
system_name	bluesky
system_machine_type	
system_arch	
system_os	
system_memory_size	
system_processor_amt	
system_l1_cache_size	
system_l2_cache_size	
system_userdata	
compiler_cpp_name	
compiler_cpp_version	
compiler_cc_name	
compiler_cc_version	
compiler_java_dirpath	
compiler_java_version	
compiler_userdata	
configure_prefix	
configure_arch	
configure_cpp	
configure_cc	
configure_jdk	
configure_profile	
configure_userdata	
userdata	

Load Trial

Trial Type: Tau profiles

Select Directory: ...

Cancel Ok

Tau profiles

Tau pprof.dat

Dynaprof

MpiP

HPCToolkit

Gprof

PSRun

ParaProf Packed Profile

Cube

HPCToolkit



Performance Database: Storage of MetaData

ParaProf Manager

File Options Help

Applications

- Standard Applications
 - Default App
 - Default Exp
 - 16pAIX200iter/s3d/taudata/rs/sameer/Users/
 - Time

Runtime Applications

DB Applications

 - AORSA2D
 - Basic run-time profiling for Socorro
 - Heap memory management for Socorro
 - hydroshock
 - MFIK

S3D

 - AIX
 - 16pAIX10iter/s3d/taudata/rs/sameer/Users/
 - 16pAIX200iter/s3d/taudata/rs/sameer/Users/
 - Time
 - 16pAIXcall200iter/s3d/taudata/rs/sameer/Users/
 - Time

Field	Value
Name	16pAIXcall200iter/s3d/taudata/rs/sameer/Users/
Application ID	8
Experiment ID	16
Trial ID	34
time	
problem_definition	nx_g=400, ny_g=400, npx=1, npx=4, npy=4, npz=1
node_count	16
contexts_per_node	1
threads_per_context	1
userdata	i_time_end=200, i_time_save=200, TAU_CALLPATH_DEPTH=2

Load Trial

Trial Type: Tau profiles

Select Directory: /Users/sameer/rs/taudata/s3d

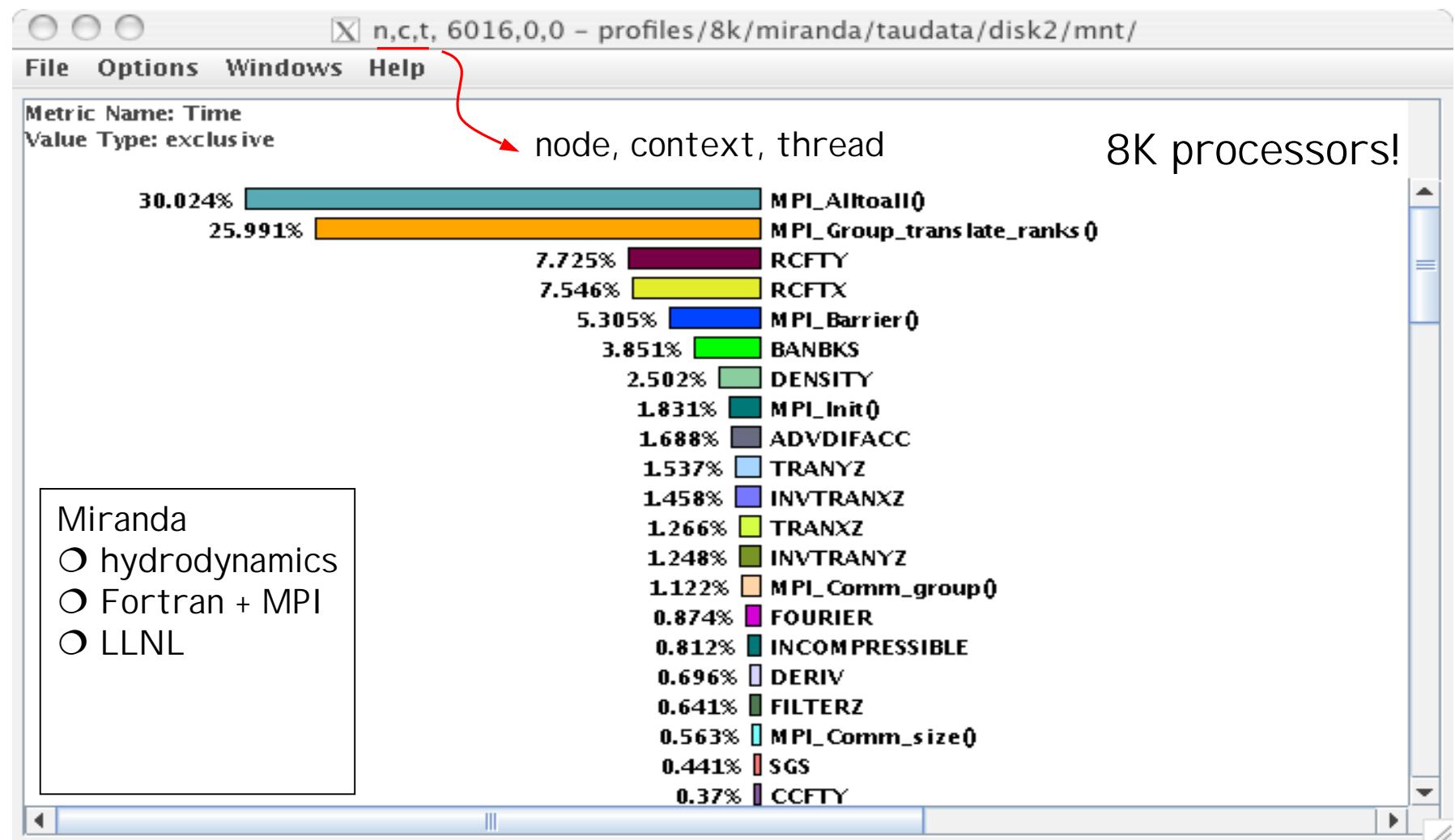
Cancel Ok



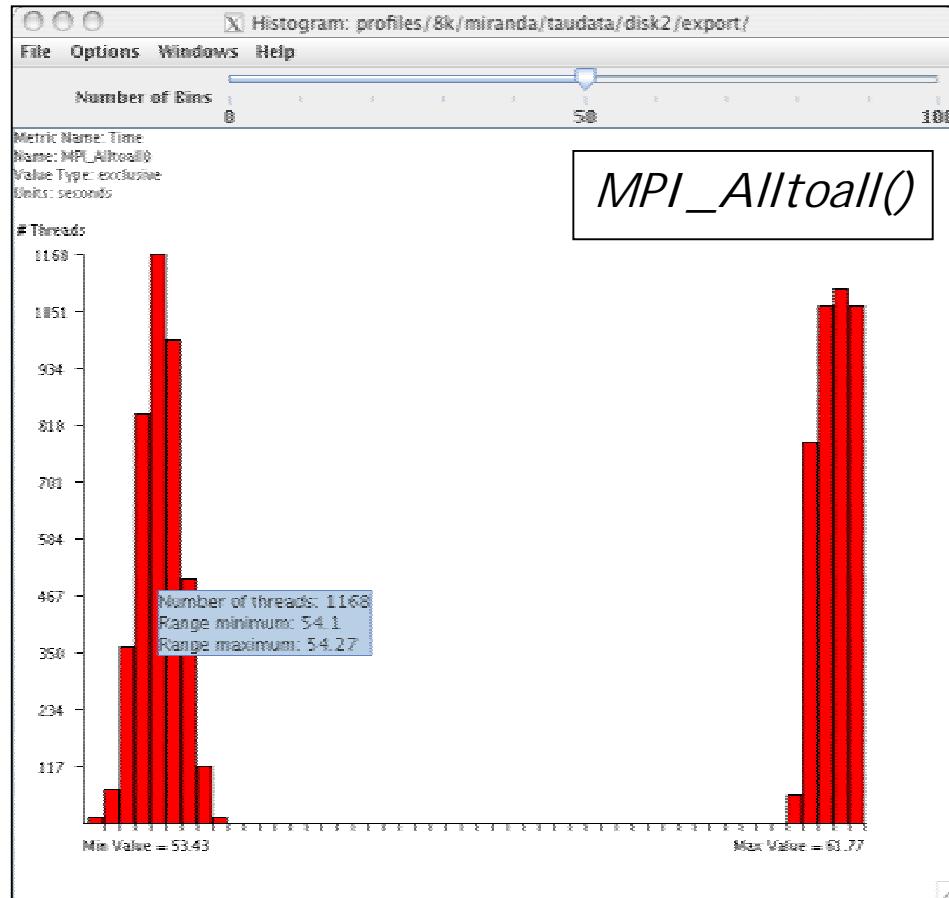
ParaProf Main Window (WRF)



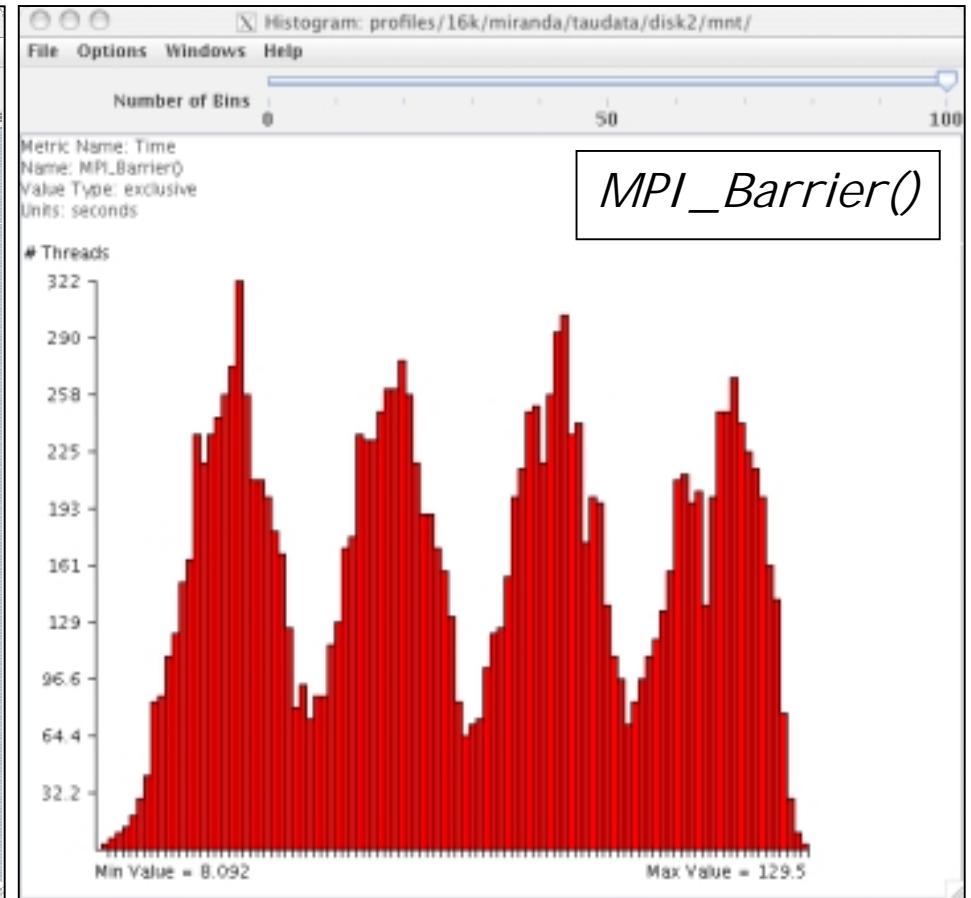
ParaProf – Flat Profile (Miranda)



ParaProf – Histogram View (Miranda)

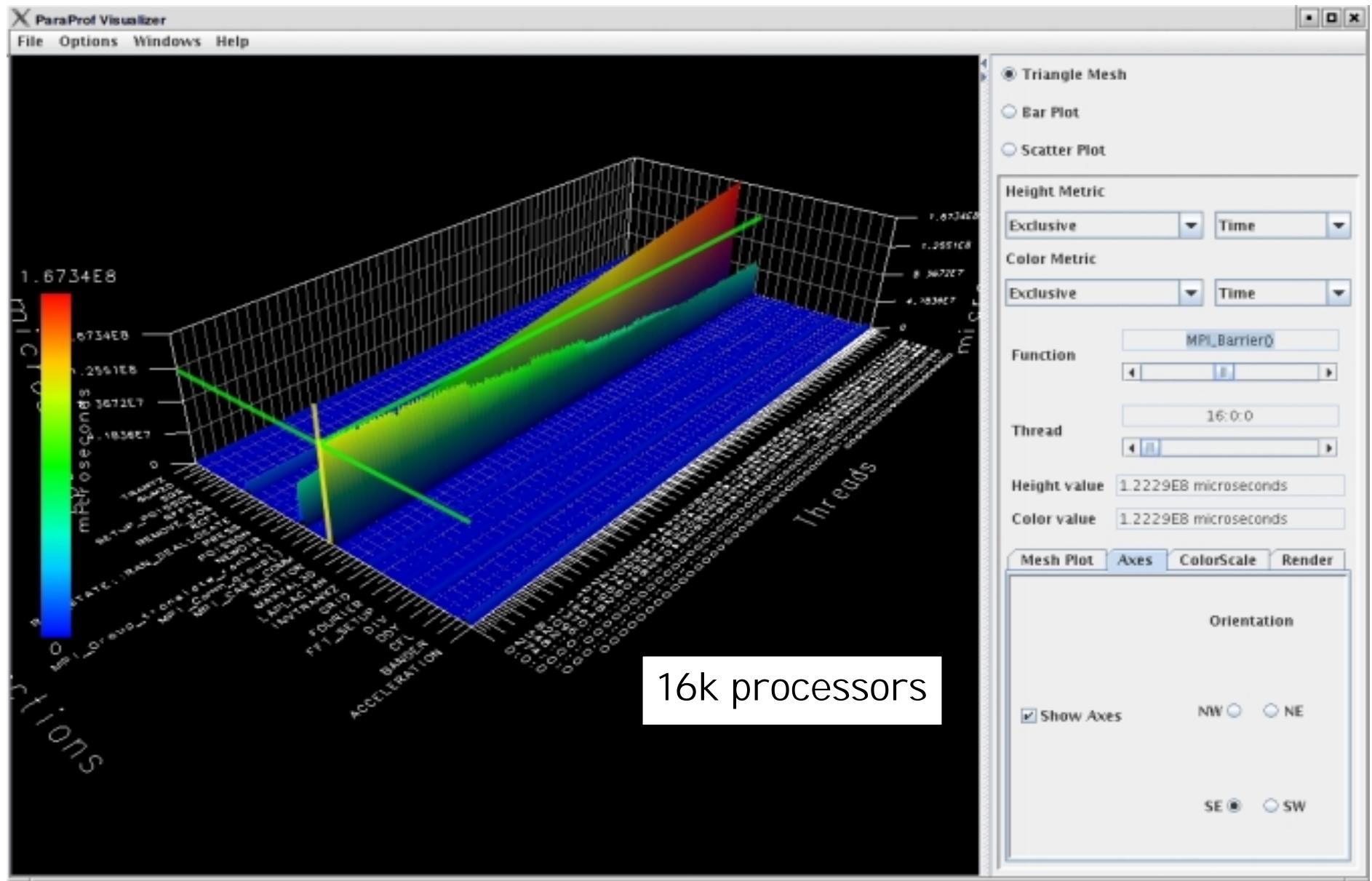


8k processors



16k processors

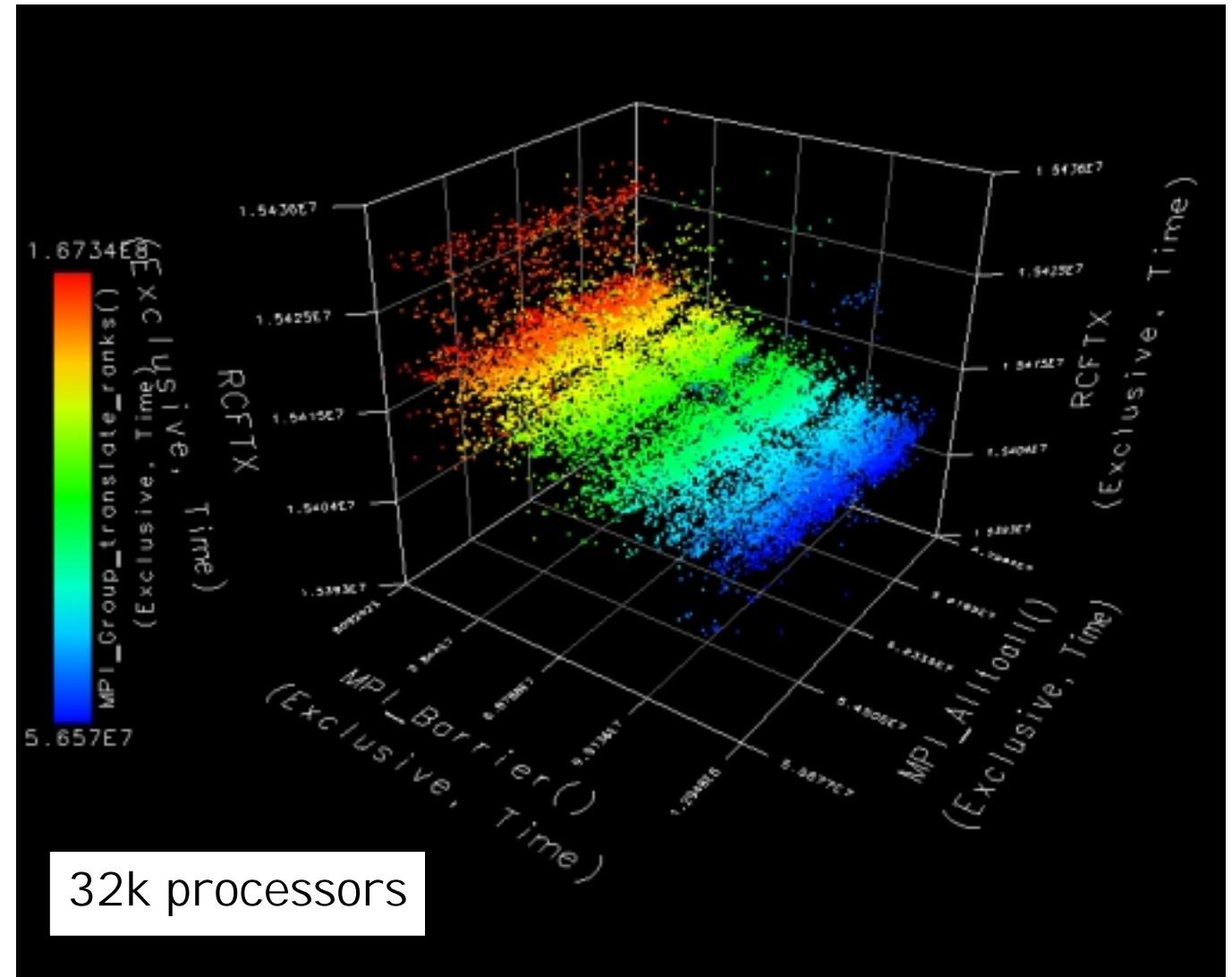
ParaProf – 3D Full Profile (Miranda)





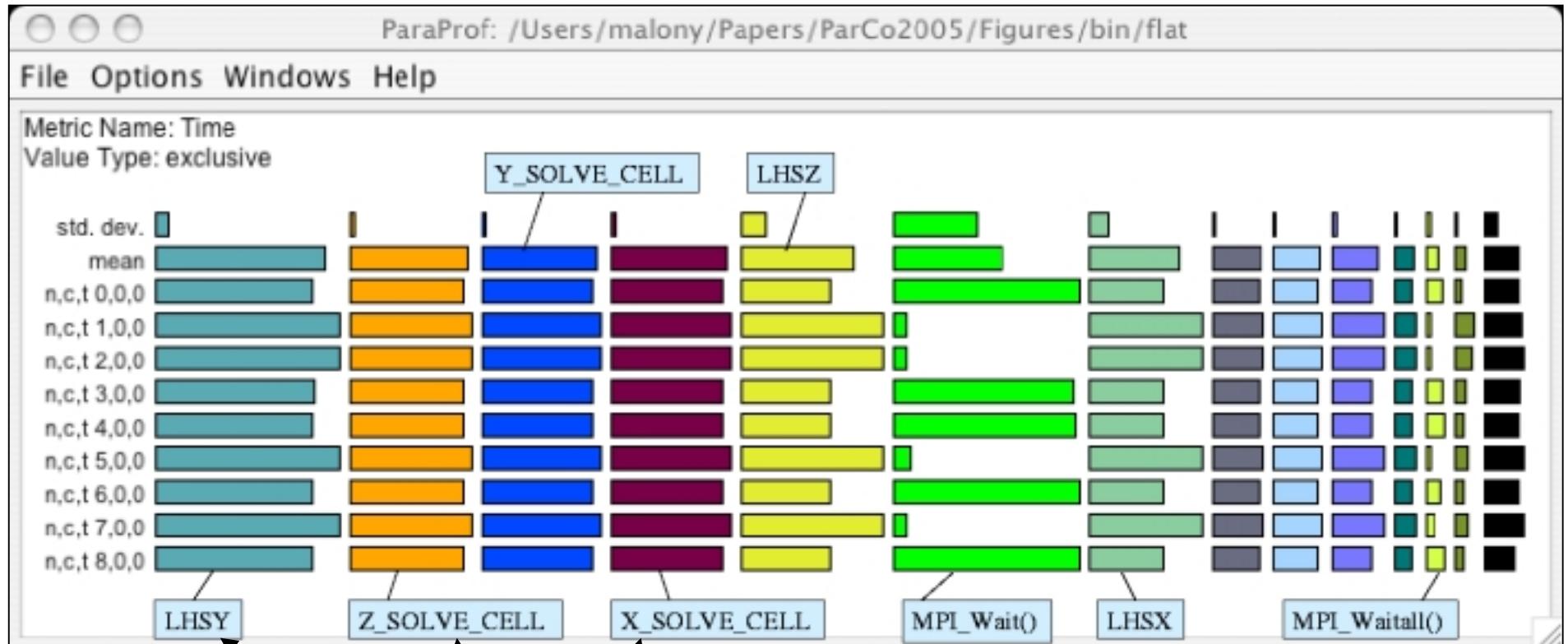
ParaProf – 3D Scatterplot (Miranda)

- Each point is a “thread” of execution
- A total of four metrics shown in relation
- ParaVis 3D profile visualization library
 - JOGL





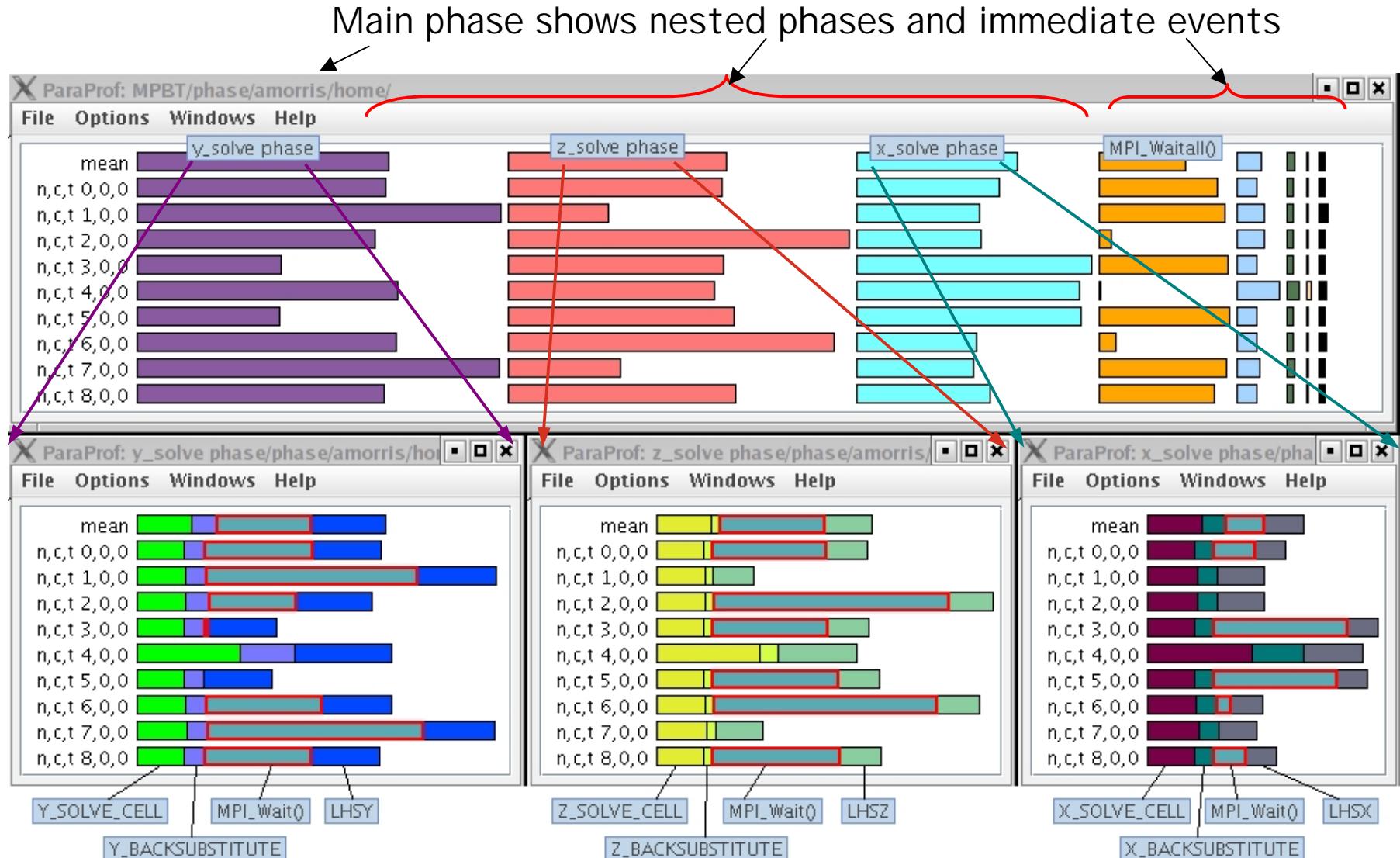
ParaProf – Flat Profile (NAS BT)



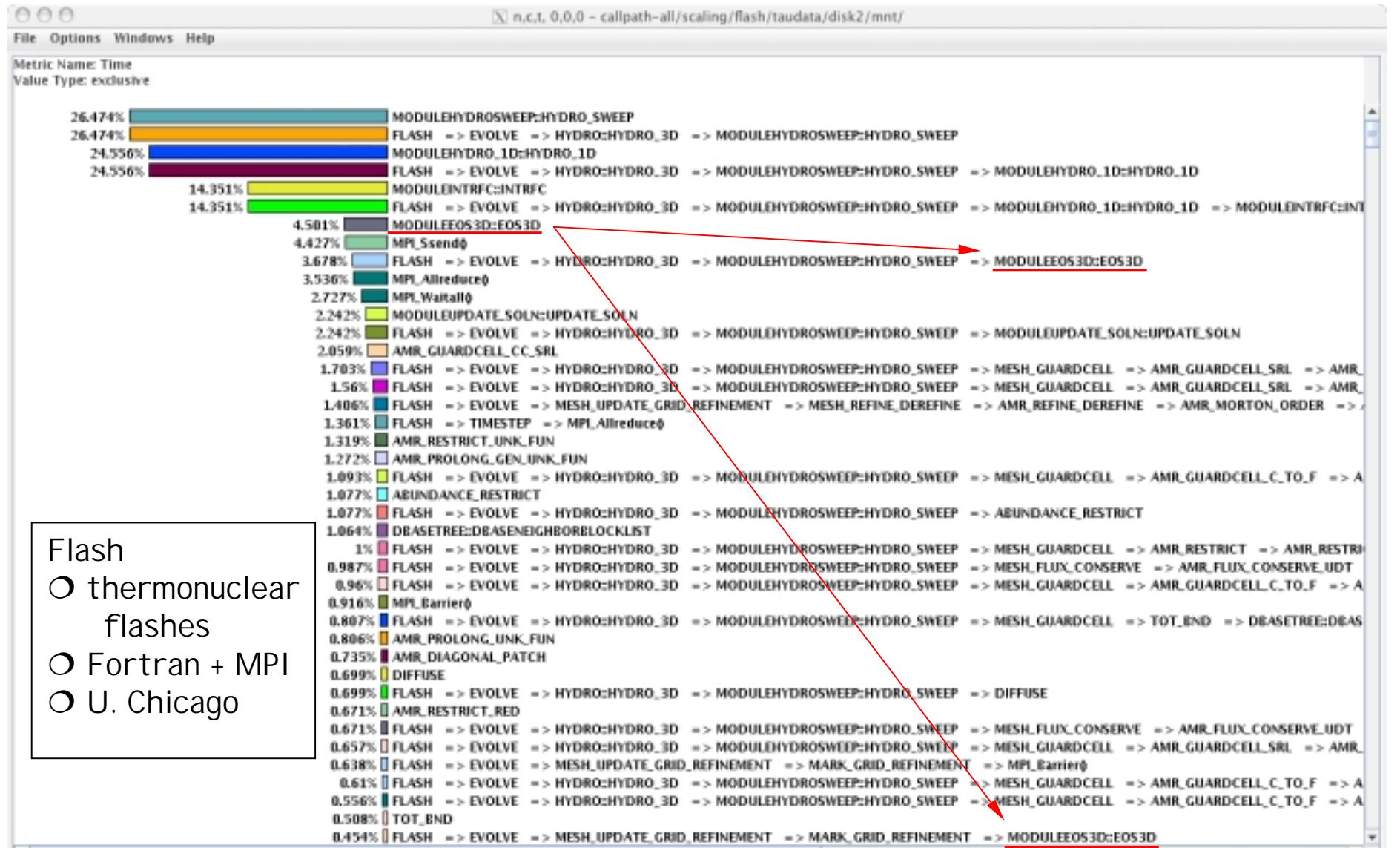
Application routine names
reflect phase semantics

How is MPI_Wait()
distributed relative to
solver direction?

ParaProf – Phase Profile (NAS BT)



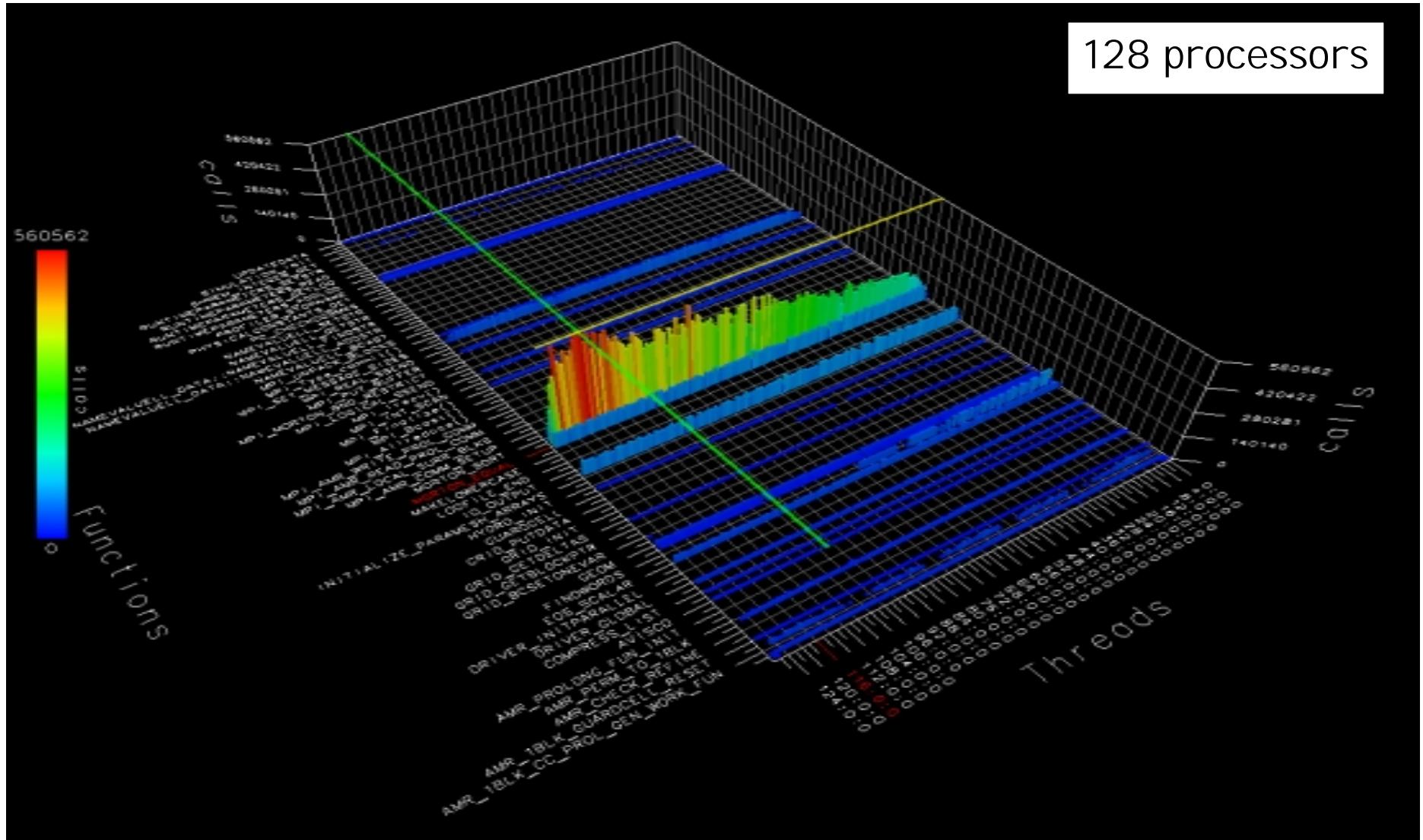
ParaProf – Callpath Profile (Flash)



Flash
 ○ thermonuclear
 flashes
 ○ Fortran + MPI
 ○ U. Chicago

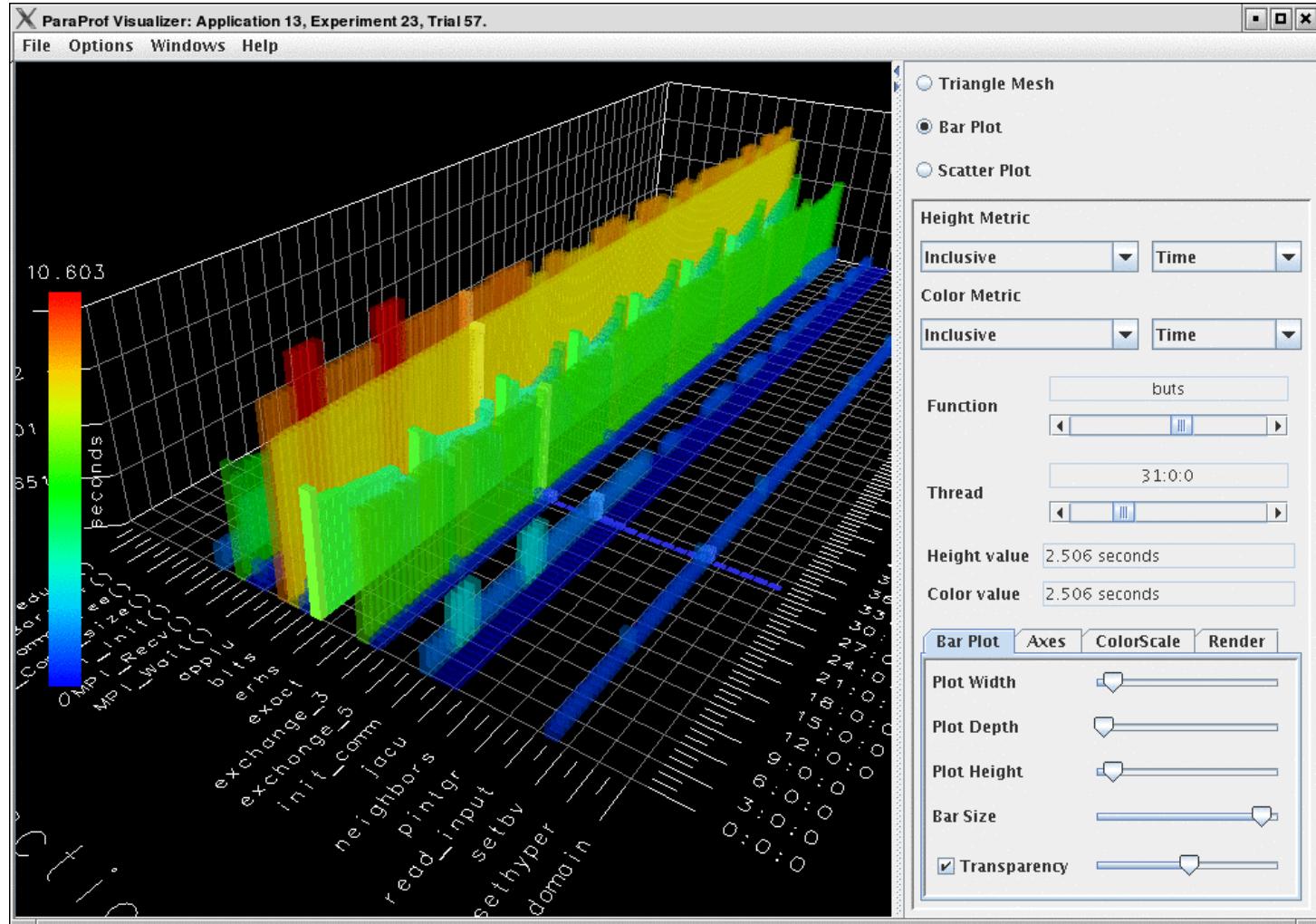


ParaProf - 3D Full Profile Bar Plot (Flash)

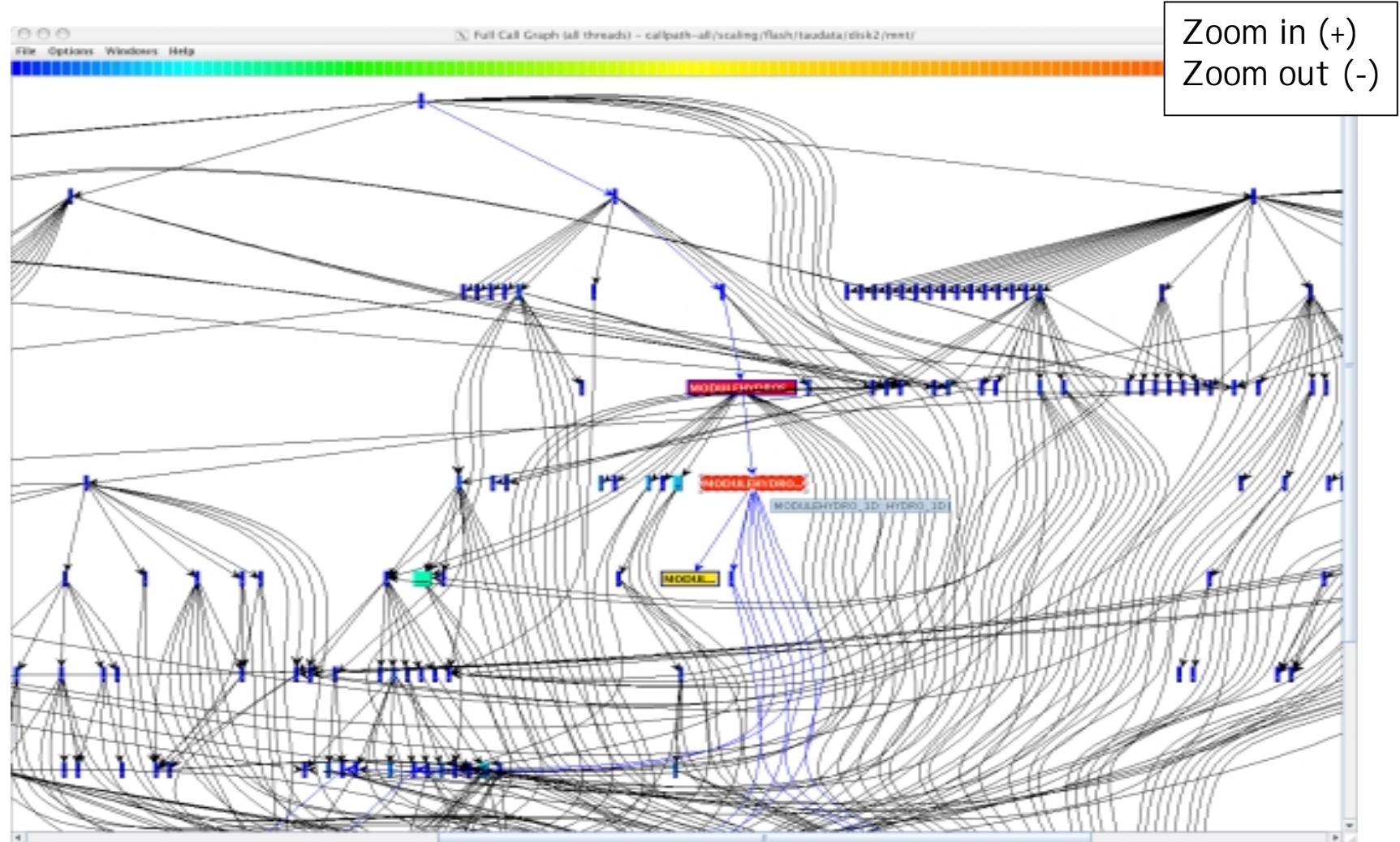




ParaProf Bar Plot (Zoom in/out +/-)



ParaProf – Callgraph Zoomed (Flash)



ParaProf - Thread Statistics Table (GSI)



Thread Statistics: n,c,t, 0,0,0 - comp.ppk/

File Options Windows Help

Name	Inclusive Time	Exclusive Time	Calls	Child Calls
GSI	5,223.564	0.098	1	30
SPECMOD::INIT_SPEC_VARS	0.26	0.26	1	0
MPI_Init()	0.056	0.054	1	1
CSISUB	5,223.094	0.012	1	13
RADINFO::RADINFO_READ	0.103	0.101	1	1,196
PCPINFO::PCPINFO_READ	0.042	0.042	1	0
GLBSOI	5,212.171	0.024	1	12
MPI_Finalize()	1.004	1.004	1	0
OBS_PARA	3.635	0.181	1	56
JFUNC::CREATE_FUNC	0.142	0.142	1	0
GUESS_GRIDDS::CREATE_GES_BIAS_GRIDS	0.059	0.059	1	0
READ_GUESS	1,406.412	0.023	1	8
READ_OBS	3,770.188	0.016	1	6
MPI_Allreduce()	3,725.802	3,725.802	3	0
READ_BUFRTOVS	44.369	0.254	1	871,535
SATTHIN::MAKEGVALS	0	0	1	0
W3FS21	0	0	1	1
BINARY_FILE.Utility::OPEN_BINARY_FILE	0.025	0.012	1	3
INITIALIZE::INITIALIZE_RTM	0.099	0.001	1	2
GUESS_GRIDDS::CREATE_SPC_GRIDS	0	0	1	0
M_FVANAGRID::ALLGETLIST_	30.582	0	1	10
ERROR_HANDLER::DISPLAY_MESSAGE	0	0	1	0
JFUNC::SET_POINTER	0	0	1	0
OZINFO::OZINFO_READ	0.016	0.016	1	0
DETER_SUBDOMAIN	0.008	0.008	1	0
GRIDMOD::CREATE_MAPPING	0.005	0.005	1	0
INIT_COMM_VARS	0.004	0.004	1	0
M_FVANAGRID::ALLGETLIST_	10.711	0	1	1
GRIDMOD::CREATE_GRID_VARS	0	0	1	0

ParaProf - Callpath Thread Relations Window

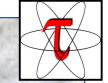


Call Path Data n,c,r, 0,0,0 – comp.ppk/

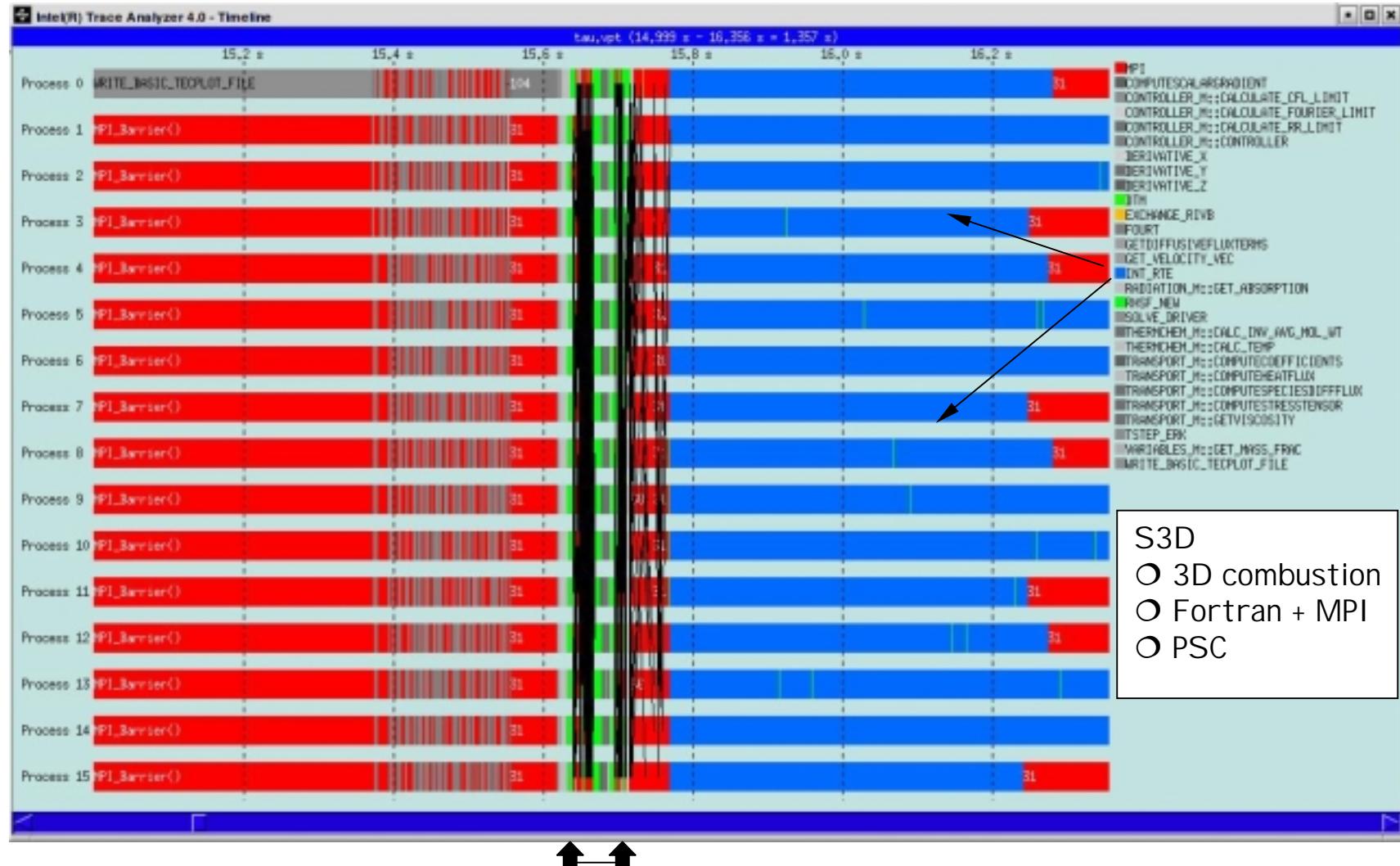
File Options Windows Help

Metric Name: Time
Sorted By: Exclusive
Units: seconds

Exclusive	Inclusive	Calls/Tot.Calls	Name[Id]
0.023	0.023	3/430	COMPUTE_DERIVED[55]
2.02	2.02	104/430	DPROCXMODD::DPROCX[66]
0.33	0.33	104/430	INSTALLMODD::INSTALL[1708]
0.003	0.003	1/430	N_PVANAGRID::ALLGETLIST_[1773]
1.639	1.639	1/430	OBS PARA[1802]
3725.802	3725.802	3/430	READ_OBS[1860]
214.294	214.294	6/430	SETUPRHSALL[1900]
20.069	20.069	208/430	STPCALCMOD::STPCALC[1942]
--> 3964.18	3964.18	430	MPI_Allreduce(){1762}
2.6E-4	30.582	1/15	GLBSCI[93]
0.007	0.036	1/15	GSI[107]
2.7E-4	10.711	1/15	GSISUB[1690]
31.273	1347.703	3/15	N_PVANAGRID::ALLGETLIST_[1773]
0.412	0.412	1/15	PRENGT[1831]
70.198	1406.389	4/15	READ_GUESS[1857]
0.952	0.952	3/15	SATTKIN::GETSPC_GLOBAL[1882]
--> 86.937	95.933	1/15	WRITE_ALL[2004]
196.61	1575.595	15	N_PVANAGRID::ALLGETLIST_[1773]
6.2E-5	6.2E-5	1/1	BALMOD::CREATE_BALANCE_VARS[7]
4.6E-5	4.6E-5	1/1	BALMOD::DESTROY_BALANCE_VARS[8]
3.494	3.494	1/1	BALMOD::PREBAL[9]
0.017	0.017	1/1	BERROR::CREATE_BERROR_VARS[11]
2.0E-4	2.0E-4	1/1	BERROR::DESTROY_BERROR_VARS[12]
8.6E-5	8.6E-5	1/1	BERROR::SET_PREDICTORS_VARS[16]
5.7E-5	5.7E-5	1/1	COMPACT_DIFFS::CREATE_CDIFF_COEPS[34]
4.9E-5	4.9E-5	1/1	COMPACT_DIFFS::DESTROY_CDIFF_COEPS[35]
0.015	0.042	1/1	COMPACT_DIFFS::ENISPH[41]
0.052	8.196	3/3	COMPUTE_DERIVED[55]
1.4E-4	3.1E-4	3/3	GETLIST_::MOVDATE_[89]
4.2E-5	4.2E-5	1/1	GRIDMOD::DESTROY_GRID_VARS[98]
8.2E-5	8.2E-5	1/1	GRIDMOD::DESTROY_MAPPING[99]
0.169	0.169	3/3	GUESS_GRIDS::CREATE_ATM_GRIDS[1692]
3.3E-4	3.3E-4	3/3	GUESS_GRIDS::DESTROY_ATM_GRIDS[1695]
9.1E-5	9.1E-5	1/1	GUESS_GRIDS::DESTROY_GRS_BIAS_GRIDS[1696]
2.2E-4	2.2E-4	1/1	GUESS_GRIDS::DESTROY_SPC_GRIDS[1697]
6.6E-5	6.6E-5	1/1	INITIALIZE::DESTROY_RTM[1705]
5.8E-5	5.8E-5	1/1	JFUNC::DESTROY_JFUNC[1739]
0.003	0.003	1/430	MPI_Allreduce(){1762}
0.017	0.017	68/116	MPI_Bcast(){1764}
0.004	0.004	297/409	MPI_Comm_rank(){1765}



Vampir – Trace Analysis (TAU-to-VTF3) (S3D)





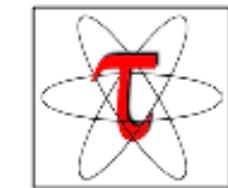
Vampir – Trace Zoomed (S3D)



PerfDMF: Performance Data Mgmt. Framework



TAU Performance System



raw profiles

* gprof
* mpiP
* psrun
* HPMtoolkit
* ...

XML document

formatted profile data

profile metadata

Performance Analysis Programs

scalability analysis

ParaProf

cluster analysis



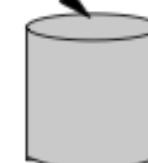
Data Mining
(Weka)

Statistics
(R / Omega)

Query and Analysis Toolkit

Java PerfDMF API

SQL (PostgreSQL, MySQL, DB2, Oracle)

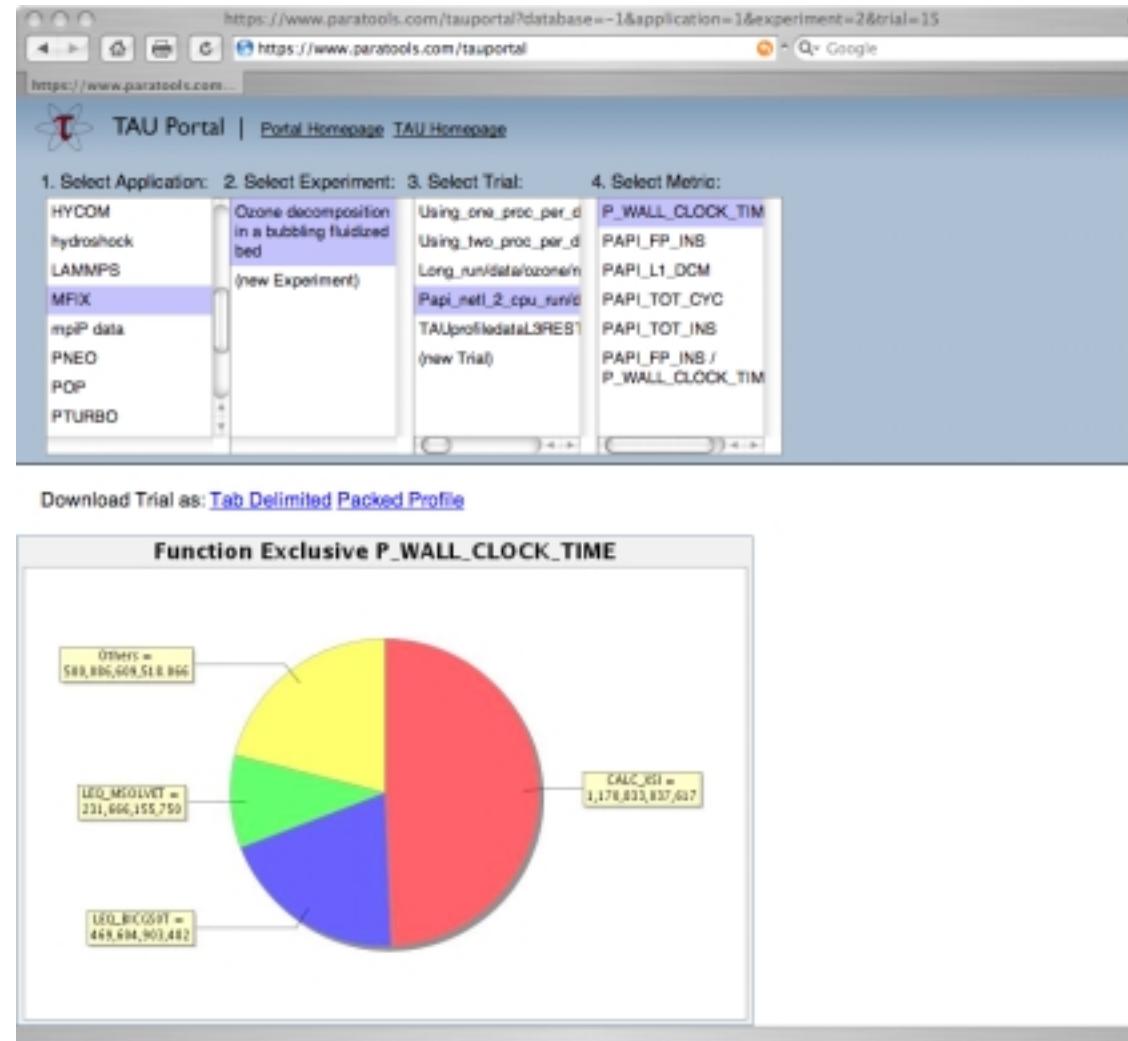


TAU Portal



The screenshot shows a web browser window for the TAU Portal at <https://www.paratools.com/tauportal>. The page features a large TAU logo and the text "TAU Portal". It includes sections for "Connect to Database" and "PerfDMF Configuration File". A file upload input shows "no file selected". Below this is a "Load Database" button. The "Connect to Database | Database Location" section contains fields for "Database type" (set to "postgresql"), "Database host name", "Database port number", "Database name", "Database username", and "User Password". There is also a checkbox for "Remember this Database" and another "Load Database" button. A note at the bottom states: "This portal is an extension of the [TAU performance system](#) and [PanaTools Inc.](#). It will connect to any database created with [perfDMF](#) using the default database schema. Within the portal you can view performance information from any trials loaded into the database. You may also interact with the database by uploading or downloading trials. This application is still under development, please send bugs or suggestion for improvements to tau [dash] bugs [at] cs.uoregon.edu." The footer includes copyright information and acknowledgments to the University of Oregon, Advanced Computing Laboratory, LANL, NM, and Research Centre Jülich, ZAM, Germany.

TAU Portal





Using Performance Database (PerfDMF)

Configure PerfDMF (Done by each user)

% perfmf_configure

- Choose derby, PostgreSQL, MySQL, Oracle or DB2
- Hostname
- Username
- Password
- Say yes to downloading required drivers (we are not allowed to distribute these)
- Stores parameters in your ~/.ParaProf/perfmdmf.cfg file

Configure PerfExplorer (Done by each user)

% perfexplorer_configure

Execute PerfExplorer

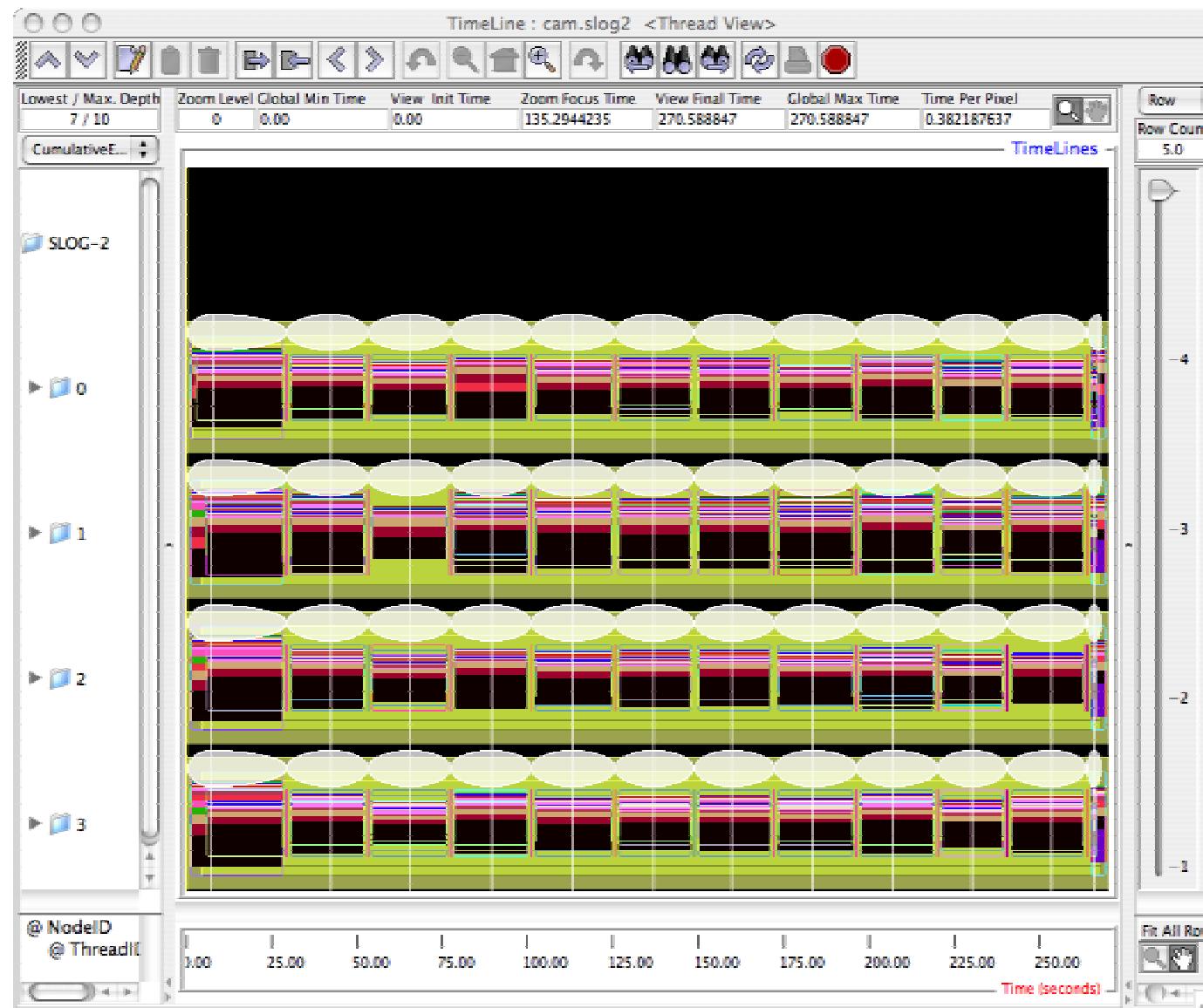
% perfexplorer



Jumpshot

- <http://www-unix.mcs.anl.gov/perfvis/software/viewers/index.htm>
- Developed at Argonne National Laboratory as part of the MPICH project
 - Also works with other MPI implementations
 - Jumpshot is bundled with the TAU package
- Java-based tracefile visualization tool for postmortem performance analysis of MPI programs
- Latest version is Jumpshot-4 for SLOG-2 format
 - Scalable level of detail support
 - Timeline and histogram views
 - Scrolling and zooming
 - Search/scan facility
- To install Jumpshot, configure TAU with -slog2 option:
`% configure -slog2 -mpi -c++=xlc_r -cc=xlc_r -mpi -pdt=<dir>`

Jumpshot





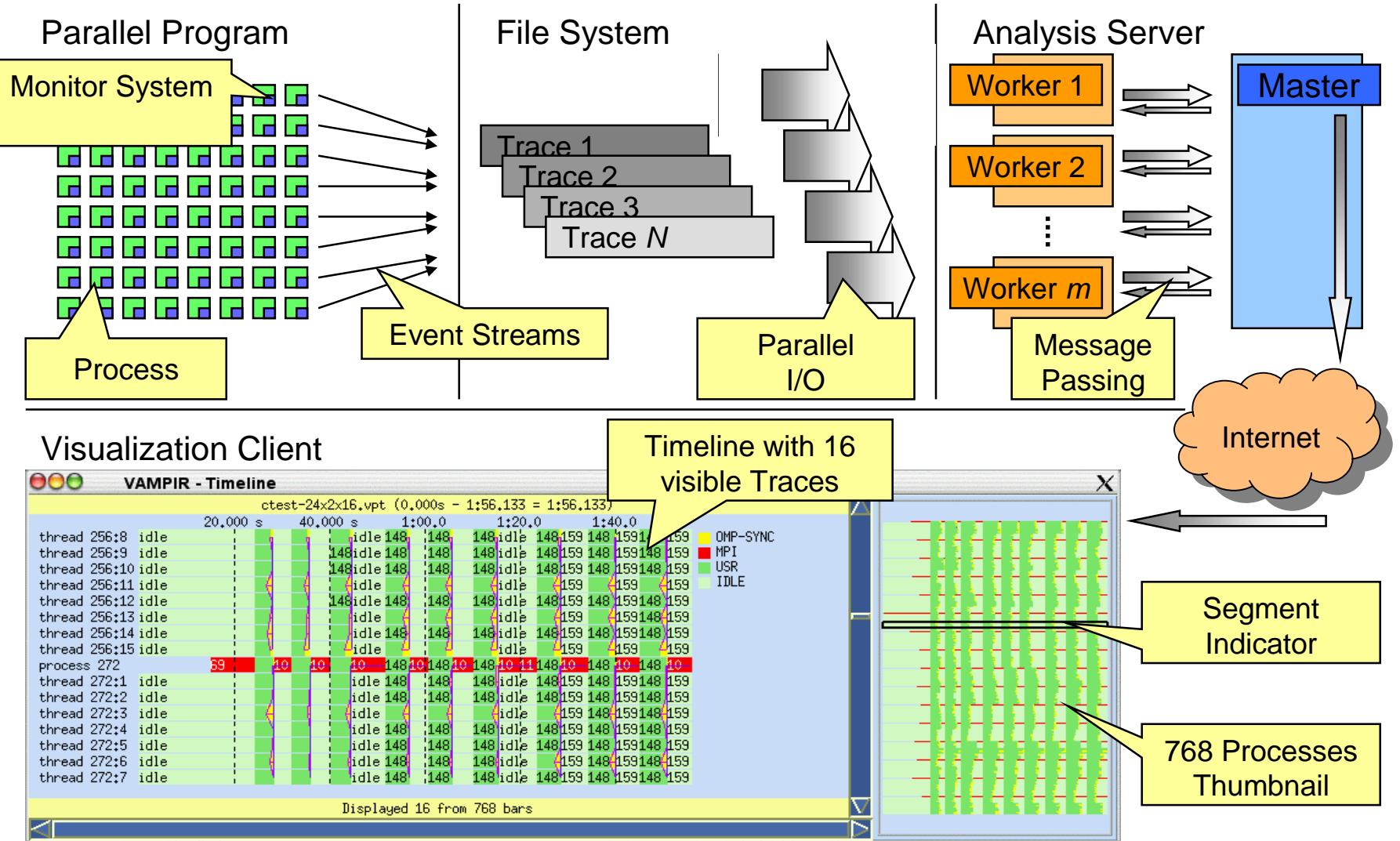
Vampir, VNG, and OTF

- Commercial trace based tools developed at ZiH, T.U. Dresden
 - Wolfgang Nagel, Holger Brunst and others...
- Vampir Trace Visualizer (aka Intel ® Trace Analyzer v4.0)
 - Sequential program
- Vampir Next Generation (VNG)
 - Client (vng) runs on a desktop, server (vngd) on a cluster
 - Parallel trace analysis
 - Orders of magnitude bigger traces (more memory)
 - State of the art in parallel trace visualization
- Open Trace Format (OTF)
 - Hierarchical trace format, efficient streams based parallel access with VNFD
 - Replacement for proprietary formats such as STF
 - Tracing library available on IBM BG/L platform
- Development of OTF supported by LLNL contract

<http://www.vampir-ng.de>

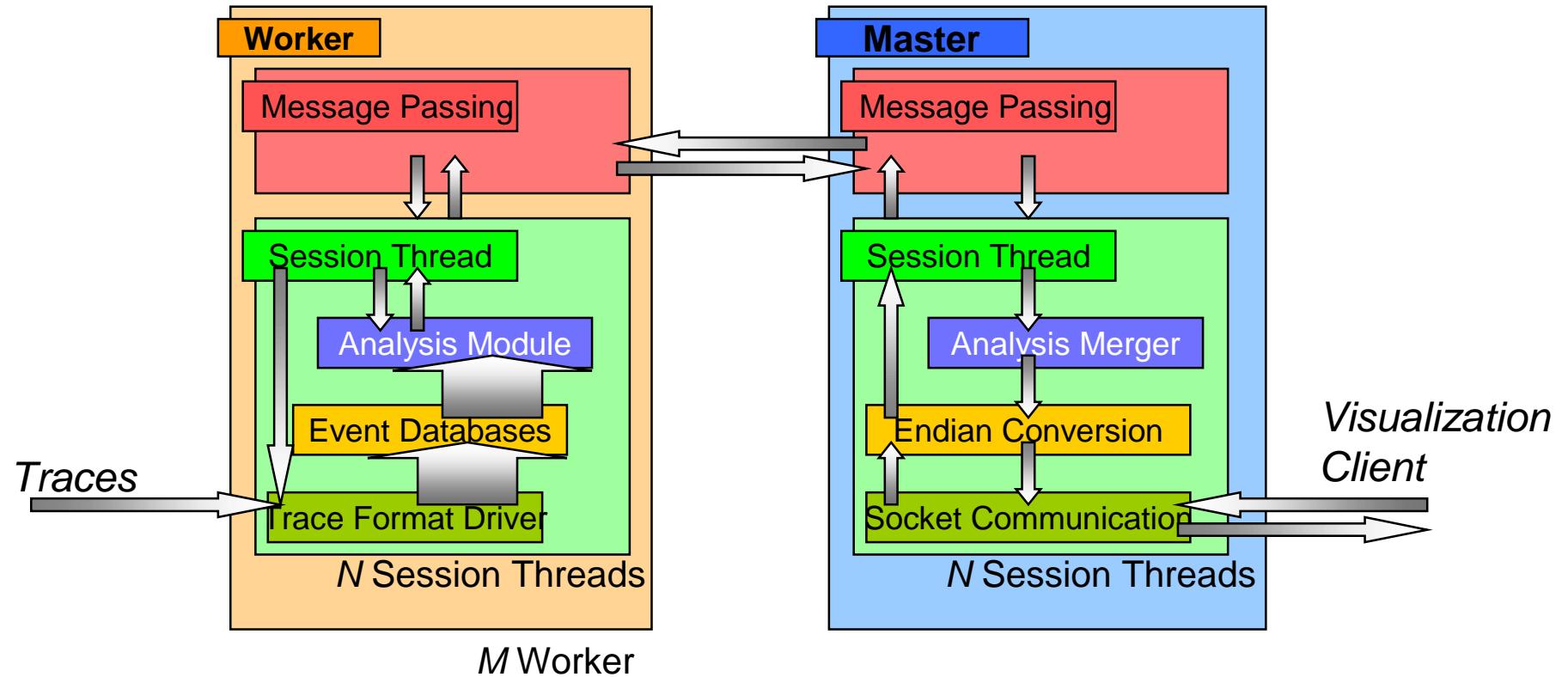


Vampir Next Generation (VNG) Architecture



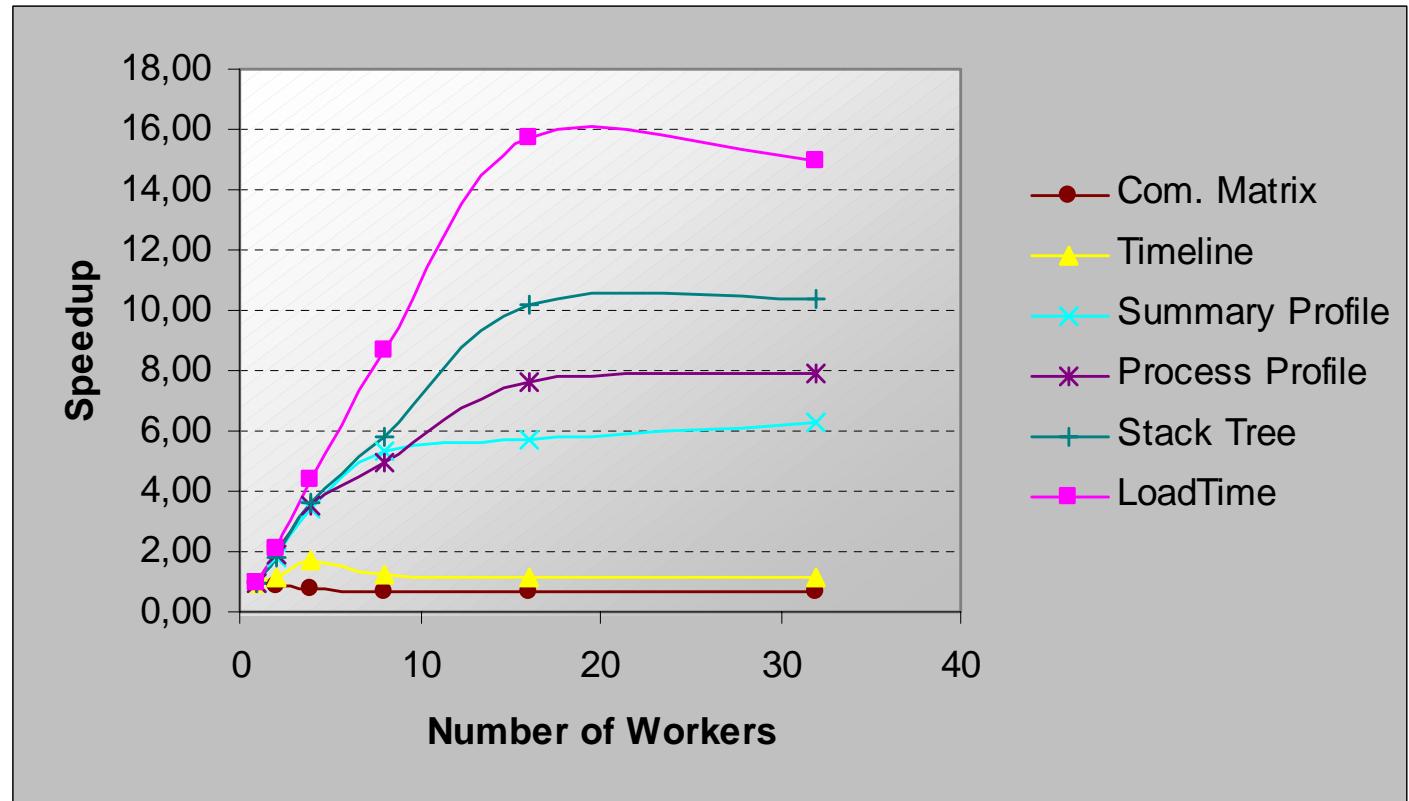


VNG Parallel Analysis Server



Scalability of VNG

- sPPM
- 16 CPUs
- 200 MB



Number of Workers	1	2	4	8	16	32
Load Time	47,33	22,48	10,80	5,43	3,01	3,16
Timeline	0,10	0,09	0,06	0,08	0,09	0,09
Summary Profile	1,59	0,87	0,47	0,30	0,28	0,25
Process Profile	1,32	0,70	0,38	0,26	0,17	0,17
Com. Matrix	0,06	0,07	0,08	0,09	0,09	0,09
Stack Tree	2,57	1,39	0,70	0,44	0,25	0,25



TAU Tracing Enhancements

- Configure TAU with **-TRACE -vtf=<dir> -otf=<dir>** options

```
% configure -TRACE -vtf=<dir> ...
```

```
% configure -TRACE -otf=<dir> ...
```

Generates tau_merge, tau2vtf, tau2otf tools in <tau>/<arch>/bin directory

```
% tau_f90.sh app.f90 -o app
```

- Instrument and execute application

```
% mpirun -np 4 app
```

- Merge and convert trace files to VTF3/SLOG2 format

```
% tau_treemerge.pl
```

```
% tau2vtf tau.trc tau.edf app.vpt.gz
```

```
% vampir foo.vpt.gz
```

OR

```
% tau2otf tau.trc tau.edf app.otf -n <numstreams>
```

```
% vampir app.otf
```

OR use VNG to analyze OTF/VTF trace files



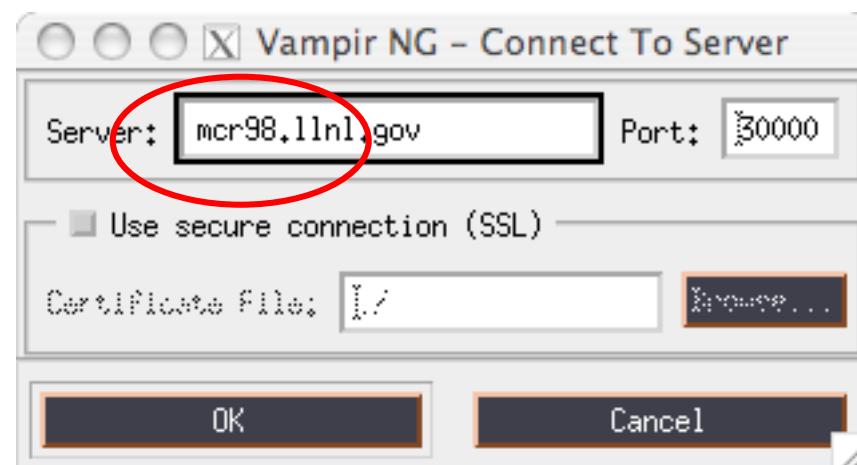
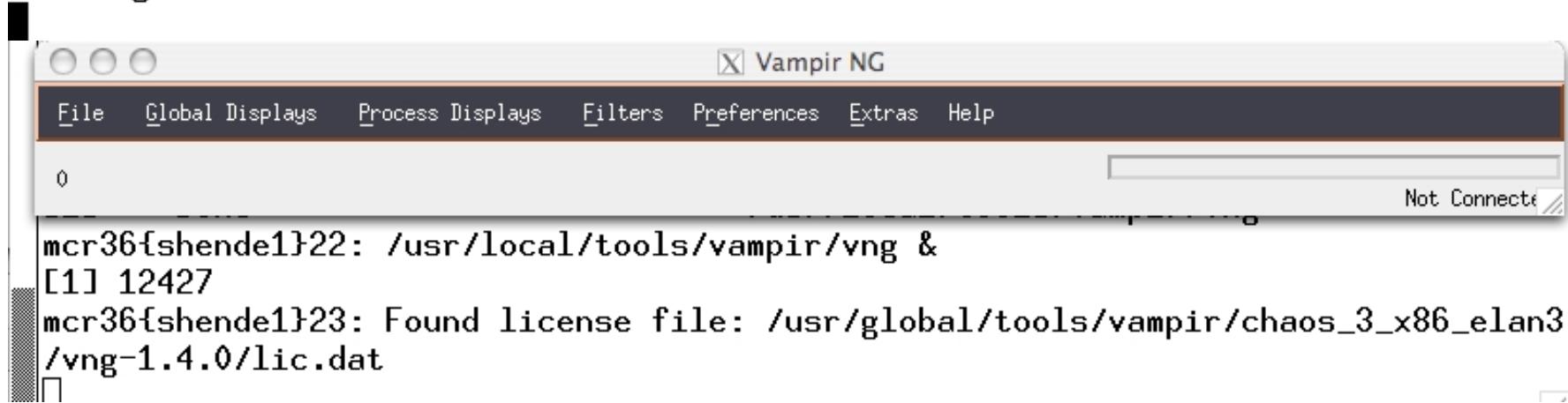
Environment Variables

- Configure TAU with -TRACE -otf=<dir> option
 - % **configure -TRACE -otf=<dir>**
 - MULTIPLECOUNTERS -papi=<dir> -mpi
 - pdt=dir ...
- Set environment variables
 - % **setenv TRACEDIR /p/gm1/<login>/traces**
 - % **setenv COUNTER1 GET_TIME_OF_DAY (reqd)**
 - % **setenv COUNTER2 PAPI_FP_INS**
 - % **setenv COUNTER3 PAPI_TOT_CYC ...**
- Execute application
 - % **poe ./a.out -procs 8**
 - % **tau_treemerge.pl** and **tau2otf/tau2vtf**



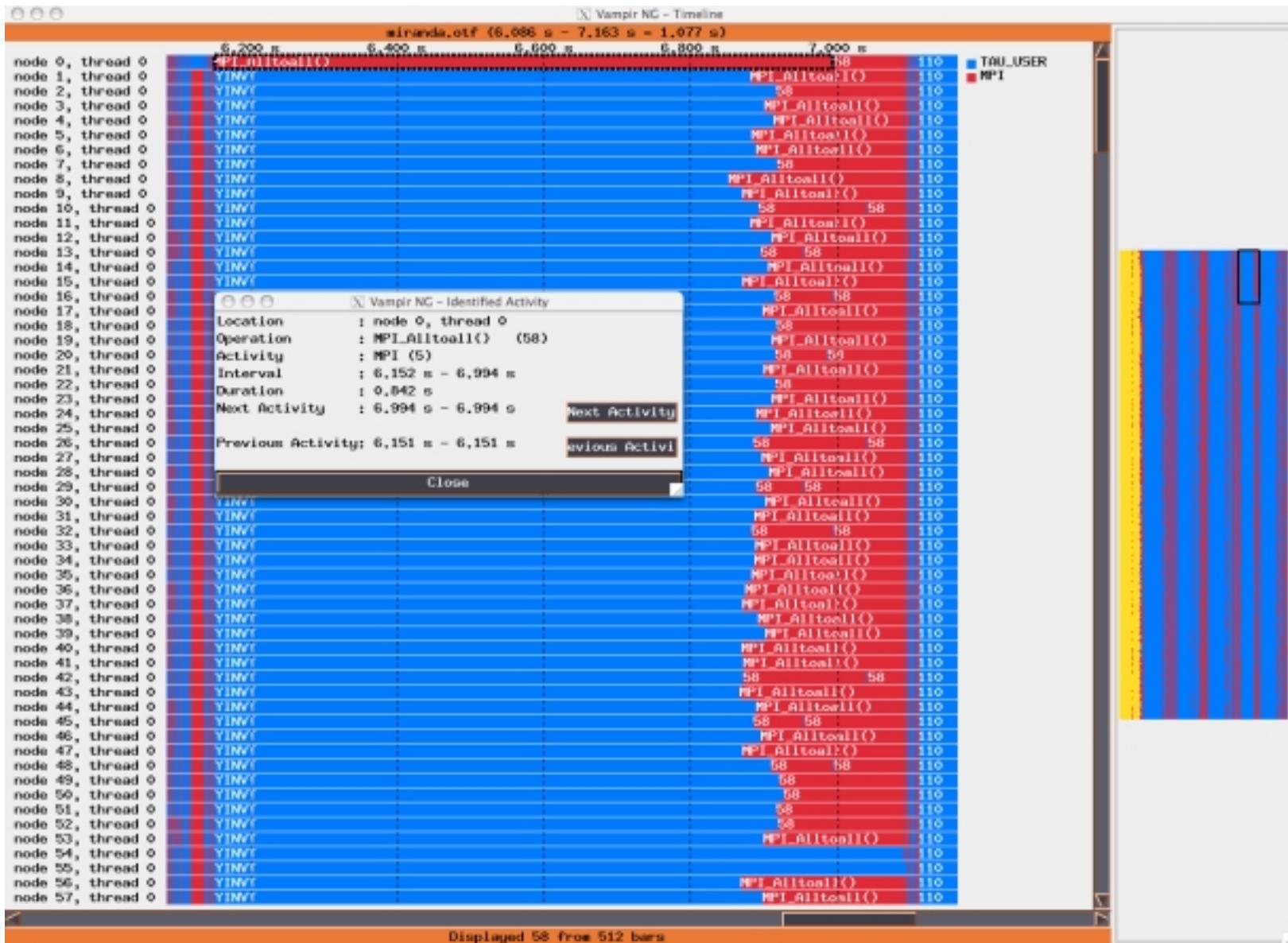
Using Vampir Next Generation (VNG v1.4)

```
mcr36{shende1}32: srun -N2 -n4 -p pdebug /usr/local/tools/vampir/vngd
Service process resides on "mcr98"
Found license file: /usr/global/tools/vampir/chaos_3_x86_elan3/vng-1.4.0/lic.dat
running...
```

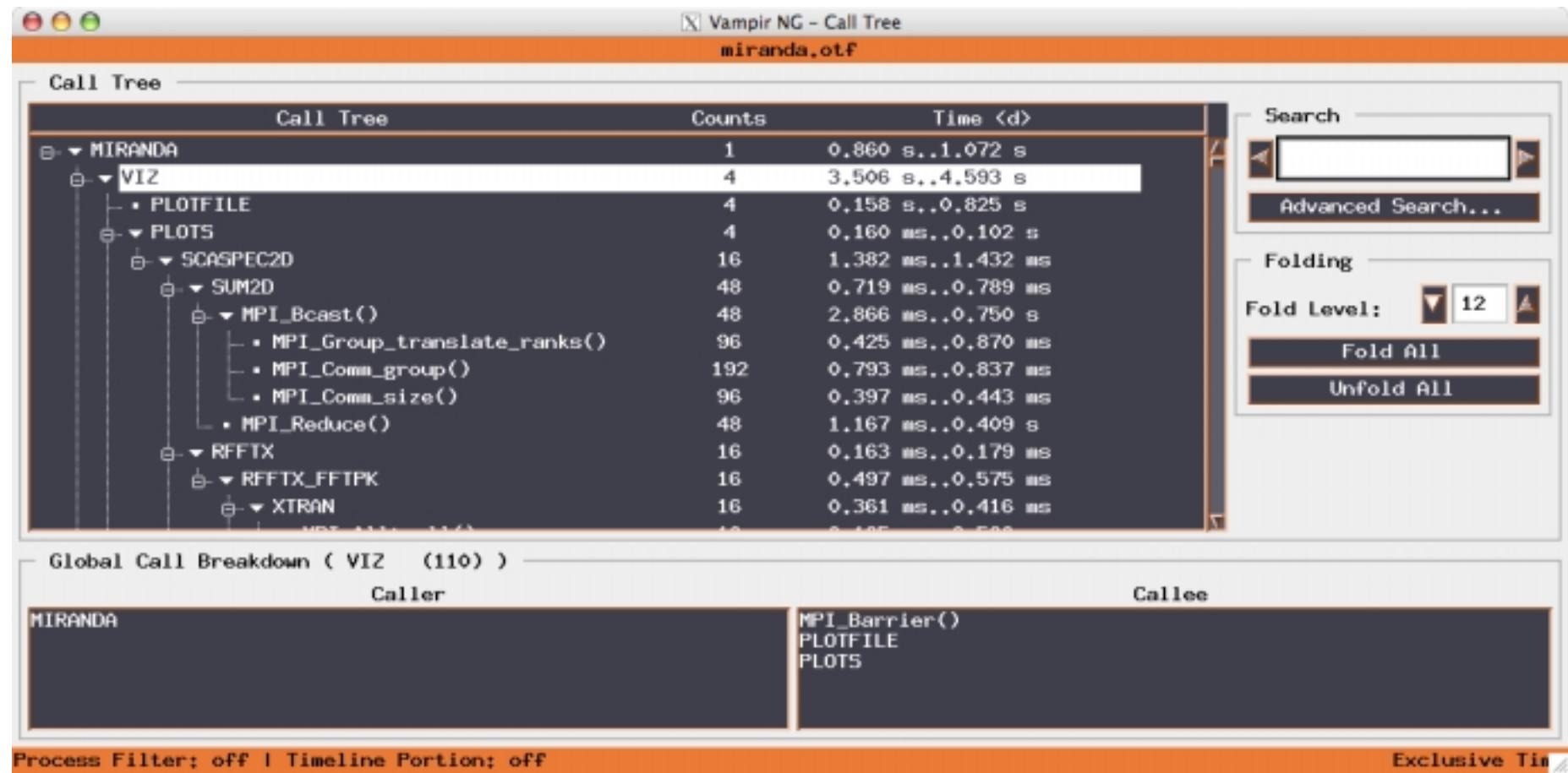




VNG Timeline Display



VNG Calltree Display

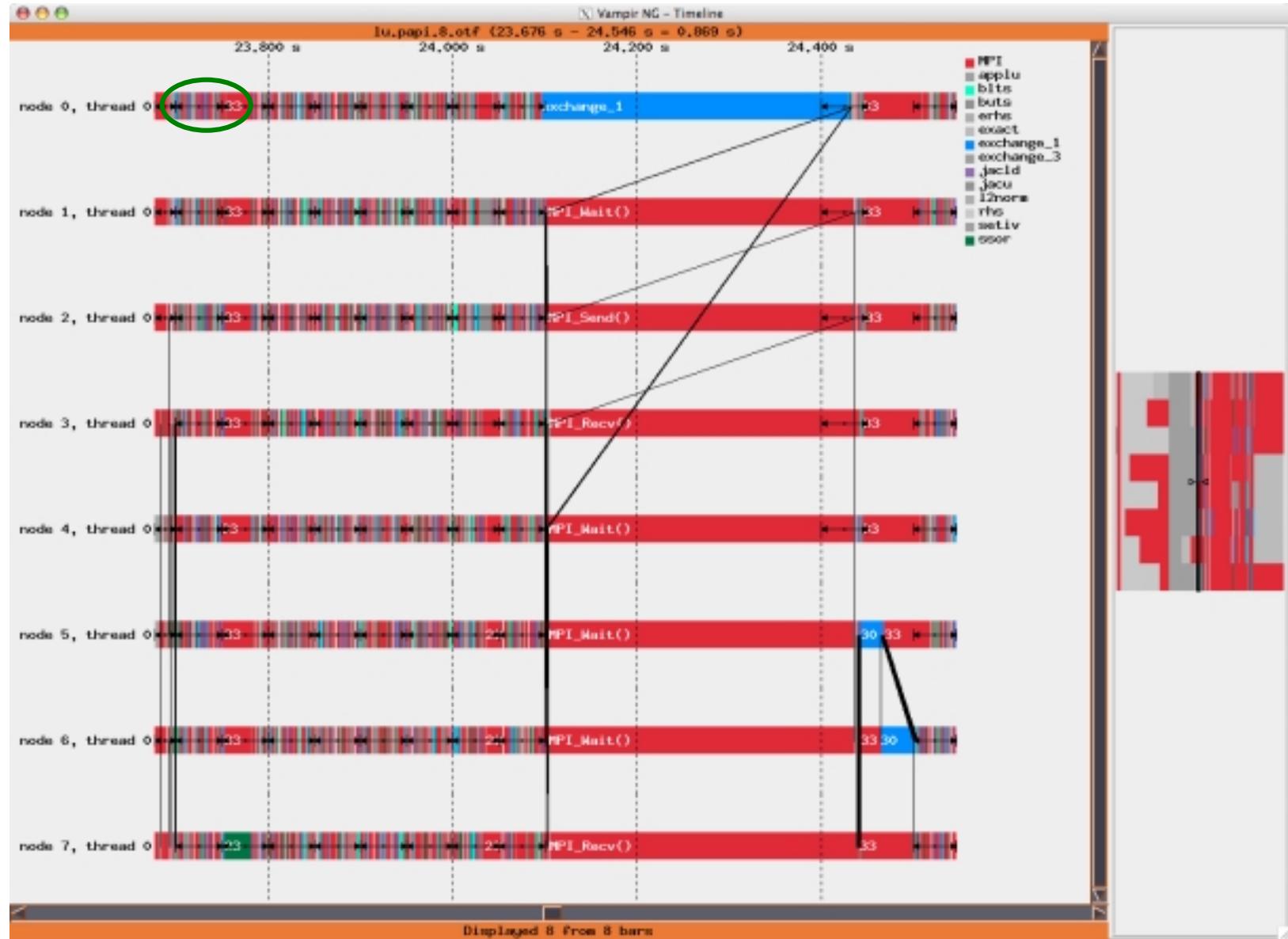




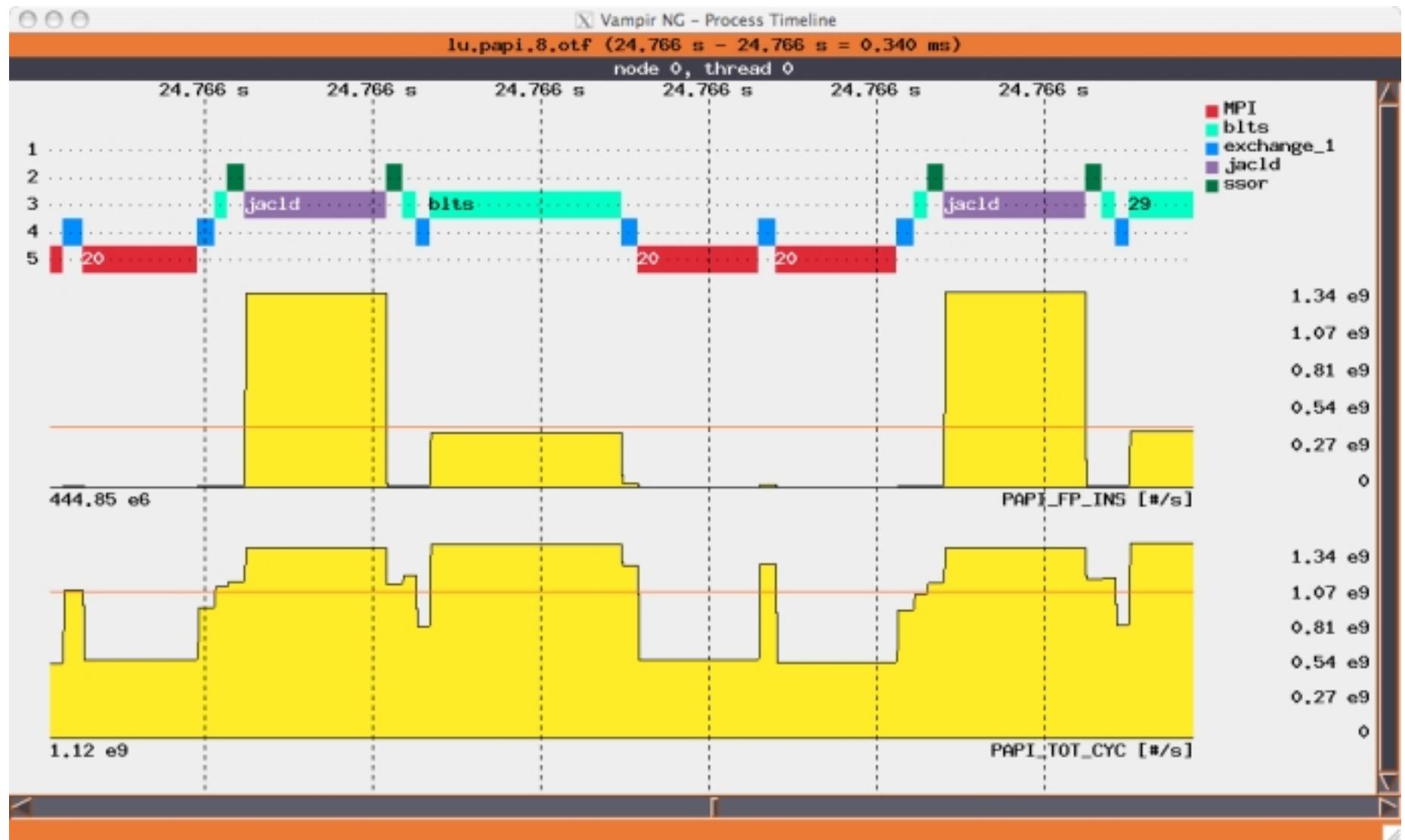
VNG Timeline Zoomed In



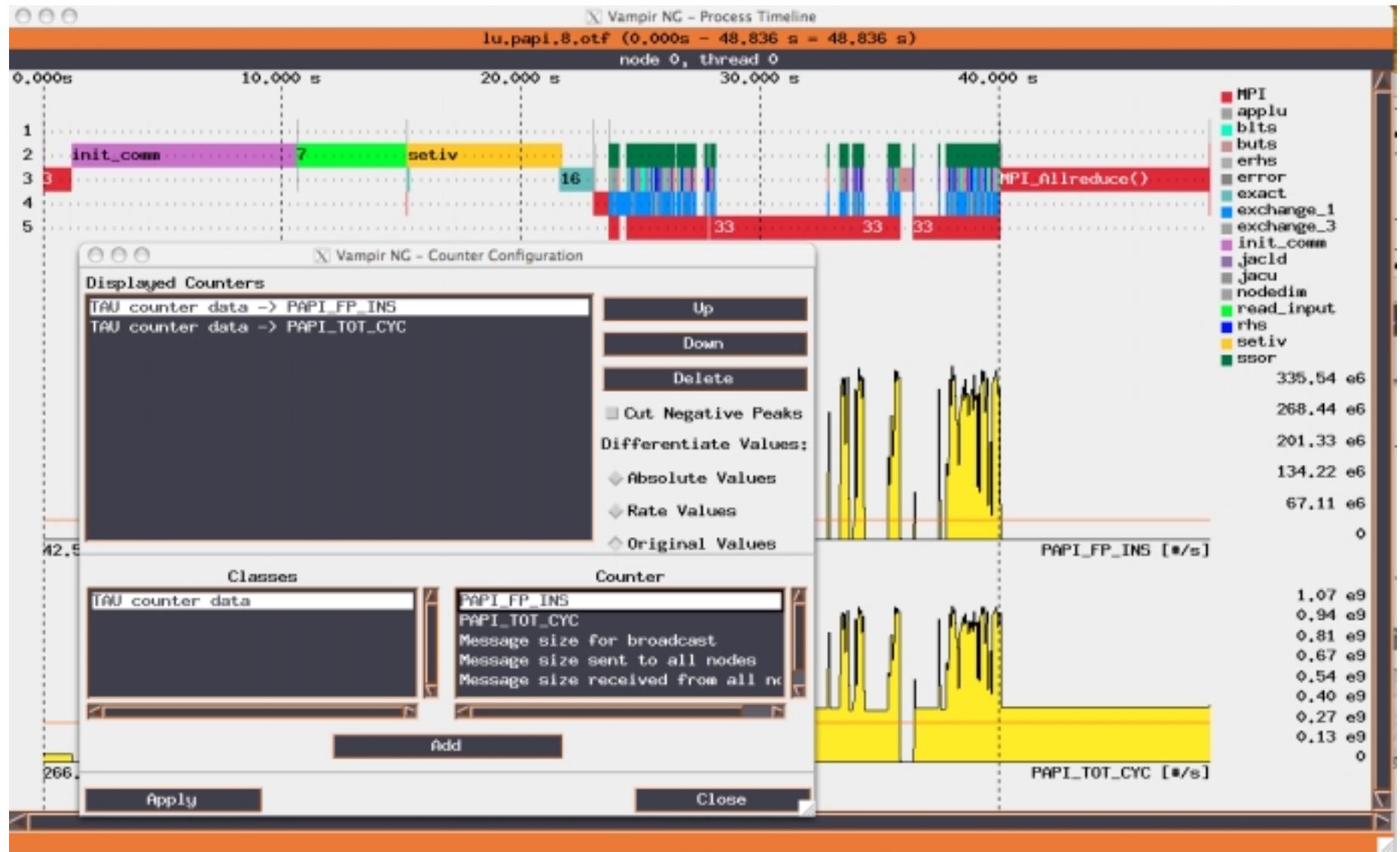
VNG Grouping of Interprocess Communications



VNG Process Timeline with PAPI Counters



OTF/VNG Support for Counters

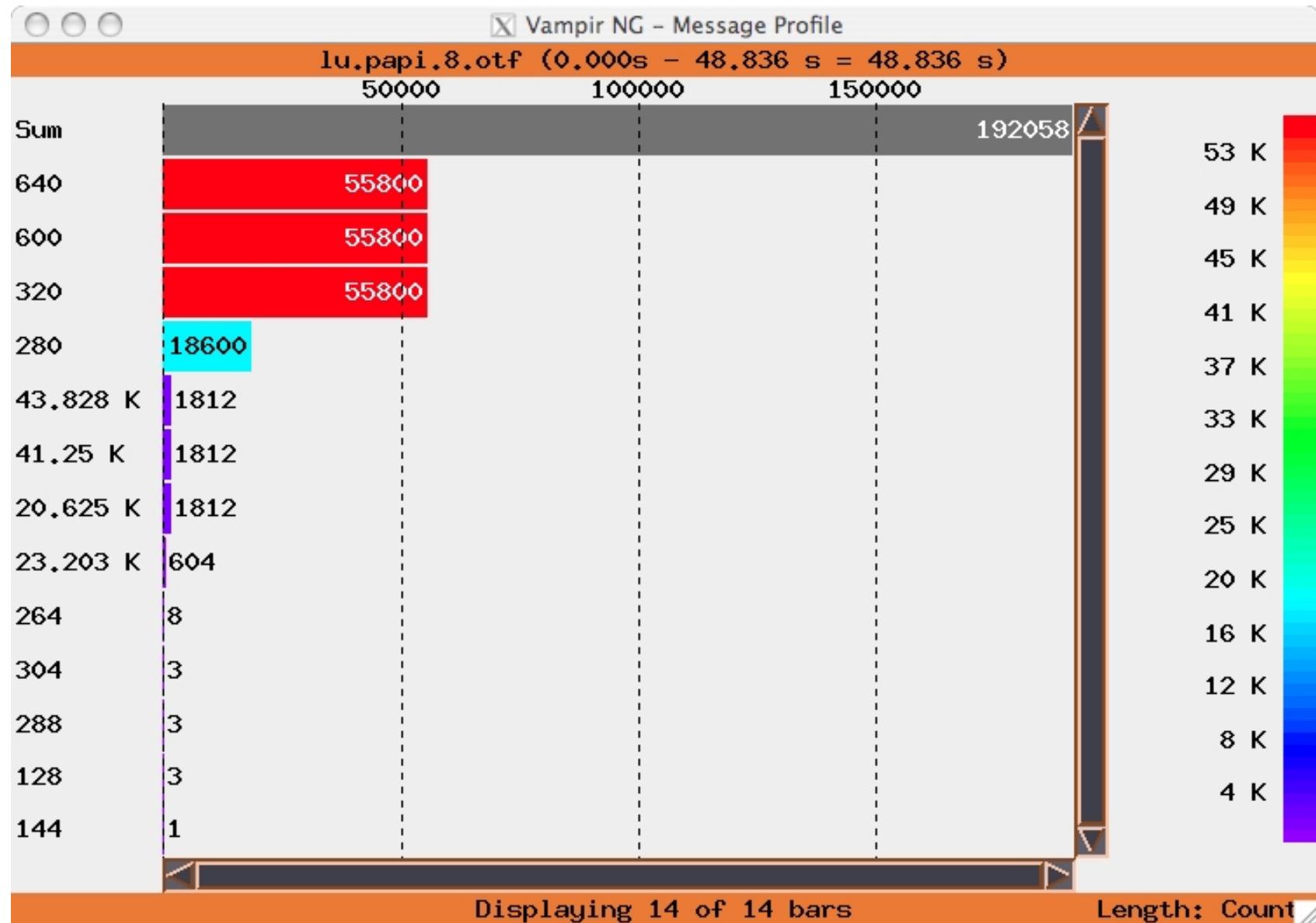




VNG Communication Matrix Display

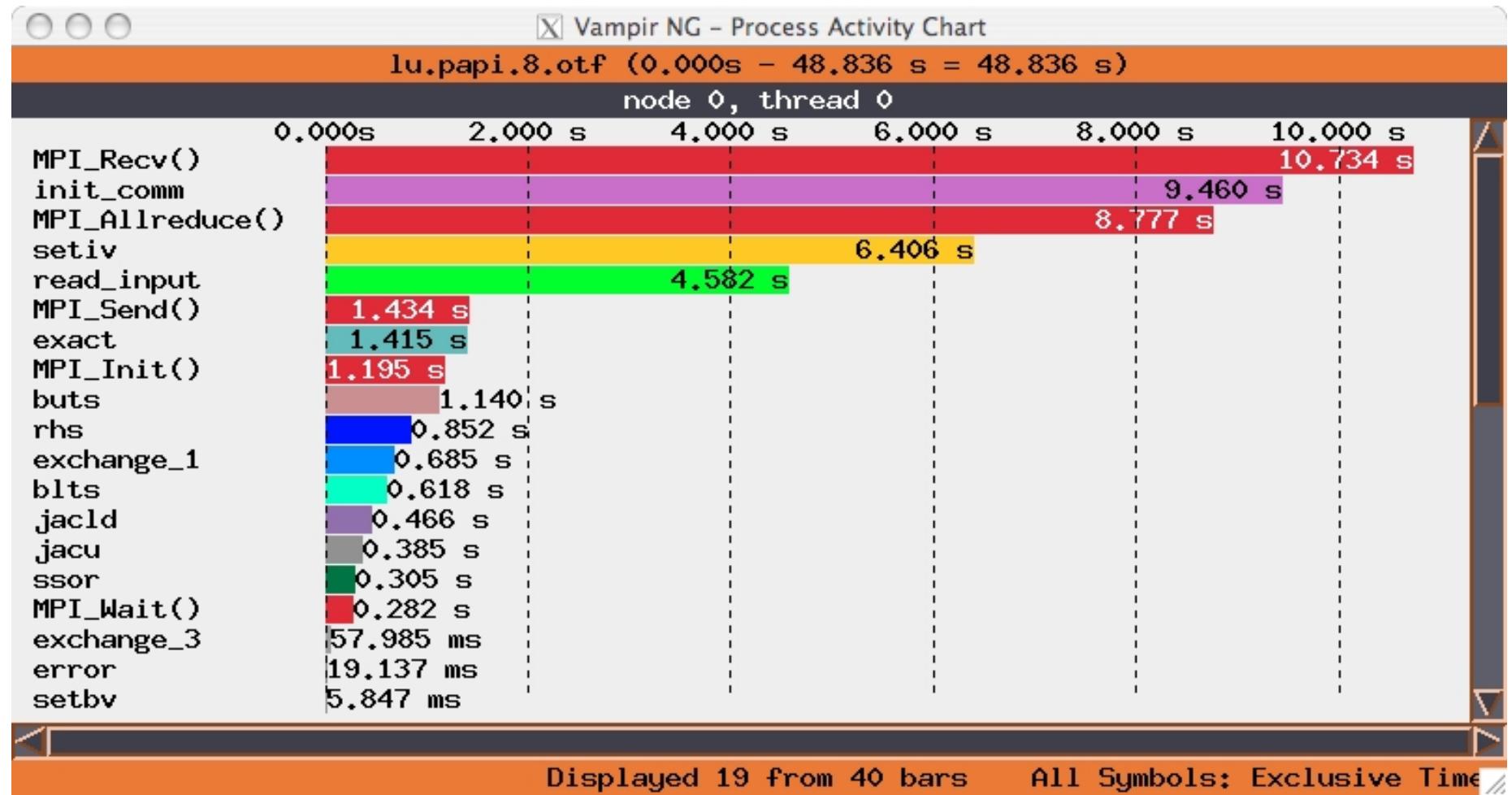


VNG Message Profile





VNG Process Activity Chart





VNG Preferences

X Vampir NG – Display Preferences

StartUp with Timeline Counter Timeline
Process Profile Summary Chart Call Tree
Message Statistics I/O Events Statistics Collective Op. Statistics

Options

Adaptive Color Legend
 Adaptive Summary Information

Display

Sum. Length Max. Length Max. Rate Max. Duration
 Counts Avg. Length Avg. Rate Avg. Duration
 Min. Length Min. Rate Min. Duration

Apply **Close**



TAU Performance System Status

- Computing platforms (selected)
 - IBM SP/pSeries/BGL, SGI Altix/Origin, Cray T3E/SV-1/X1/XT3, HP (Compaq) SC (Tru64), Sun, Linux clusters (IA-32/64, Alpha, PPC, PA-RISC, Power, Opteron), Apple (G4/5, OS X), Hitachi SR8000, NEC SX-5/6, Windows ...
- Programming languages
 - C, C++, Fortran 77/90/95, HPF, Java, Python
- Thread libraries (selected)
 - pthreads, OpenMP, SGI sproc, Java, Windows, Charm++
- Compilers (selected)
 - Intel, , GNU, Fujitsu, Sun, PathScale, SGI, Cray, IBM, HP, NEC, Absoft, Lahey, Nagware



Concluding Discussion

- Performance tools must be used effectively
- More intelligent performance systems for productive use
 - Evolve to application-specific performance technology
 - Deal with scale by “full range” performance exploration
 - Autonomic and integrated tools
 - Knowledge-based and knowledge-driven process
- Performance observation methods do not necessarily need to change in a fundamental sense
 - More automatically controlled and efficiently use
- Develop next-generation tools and deliver to community
- Open source with support by ParaTools, Inc.
- <http://www.cs.uoregon.edu/research/tau>

Labs!



Lab: TAU

Lab Instructions



Get `workshop.tar.gz` on `Seaborg.nersc.gov` using:

```
% cp /usr/common/acts/TAU/workshop.tar.gz
```

Or

```
% wget  
http://www.cs.uoregon.edu/research/tau/  
workshop.tar.gz
```

```
% gtar zxf workshop.tar.gz
```

and follow the instructions in the `README` file.

Lab Instructions



To profile a code:

1. Load TAU module:

```
% module load tau
```

2. Change the compiler name to `tau_cxx.sh`, `tau_f90.sh`, `tau_cc.sh`:

```
F90 = tau_f90.sh
```

3. Choose TAU stub makefile

```
% setenv TAU_MAKEFILE
```

```
/usr/common/acts/TAU/2.15.5/rs6000/lib/Makefile.tau-[options]
```

4. If stub makefile has `-multiplecounters` in its name, set COUNTER[1-<n>] environment variables:

```
% setenv COUNTER1 GET_TIME_OF_DAY
```

```
% setenv COUNTER2 PAPI_FP_INS
```

```
% setenv COUNTER3 PAPI_TOT_CYC ...
```

5. Set `TAU_THROTTLE` environment variable to throttle instrumentation:

```
% setenv TAU_THROTTLE 1
```

6. Build and run workshop examples, then run `pprof/paraprof`

Support Acknowledgements



- Department of Energy (DOE)
 - Office of Science MICS office contracts
 - University of Utah ASC Level 1 sub-contract
 - Lawrence Livermore National Lab contracts
 - Argonne National Laboratory FastOS contracts
 - Los Alamos National Laboratory contracts
- NSF
 - High-End Computing Grant
- T.U. Dresden, GWT
 - Dr. Wolfgang Nagel and Holger Brunst
- Research Centre Juelich
 - Dr. Bernd Mohr



ParaTools